Universidade Federal do Paraná

Katheryne Louise Graf

Análise comparativa de Softwares para Detecção de Rootkits em ambiente Linux, sob a visão do usuário

Curitiba PR
2016

Katheryne Louise Graf

Análise comparativa de Softwares para Detecção de Rootkits em ambiente Linux, sob a visão do usuário

Trabalho apresentado como requisito parcial à conclusão do Curso de Bacharelado em Ciência da Computação, setor de Ciências Exatas, da Universidade Federal do Paraná.

Área de concentração: *Ciência da Computação*.

Orientador: Carlos Alberto Maziero.

Curitiba PR
2016

Ficha Catalográfica

Esta folha deve ser substituída pela ficha catalográfica fornecida pela Biblioteca Central da UFPR, a pedido da secretaria do PPGInf/UFPR (vide arquivo `ficha.tex`*).*

Termo de aprovação

Esta folha deve ser substituída pela ata de defesa ou termo de aprovação devidamente assinado, que será fornecido pela secretaria do programa após a defesa ter sido concluída e aprovada (vide arquivo `aprovacao.tex`*).*

"É sábio reconhecer a necessidade, quando todas as outras soluções já foram ponderadas, embora possa parecer tolice para aqueles que têm falsas esperanças."

J.R.R. Tolkien

Agradecimentos

Primeiramente devo agradecer a Deus, por ter me escutado em todas as horas e momentos em que precisei que um auxílio para continuar, por todas as oportunidades que obtive até aqui, por todos os obstáculos que apareceram em meu caminho e que eu consegui transpor.

Agradeço a Universidade Federal do Paraná, por me dar a oportunidade de fazer parte de seu corpo de alunos, poder me graduar no curso de Ciência da Computação e fazer parte de uma pequena parcela da sua história.

Agradeço à todos os meus mestres que, cada uma a sua maneira, conseguiram me transmitir seus conhecimentos, me fizeram admirá-los por suas capacidades e força de vontade em estarem sempre melhorando e aprendendo novas formas para transmitir aos seus alunos o seu conhecimento. Agradeço em especial ao meu orientador, Professor Doutor Carlos Alberto Maziero, por ter aceitado o meu convite, pelo apoio, dedicação e paciência na elaboração deste trabalho.

Agradeço a minha família, em especial meus pais, que sempre me apoiaram, incentivaram e me ajudaram a poder realizar este sonho que é o de fazer uma faculdade; minha irmã Manuella e meu cunhado Adriano, por todo apoio, incentivo e por nunca me deixarem desistir.

Agradeço aos meus amigos por compreenderem a minha ausência necessária, paciência nos momentos difíceis, por sempre me ajudarem quando precisei, por todas as risadas nas situações de aflição, pelos abraços nos momentos de angústia.

A todos que direta ou indiretamente fizeram parte da minha formação, o meu muito obrigado.

Resumo

O mundo computacional mudou, as tecnologias se desenvolveram em âmbito de software e hardware, mas assim com surgiram novas aplicações benignas e outras foram melhoradas, o mesmo aconteceu com as aplicações malignas. *Rootkits* deveriam ser softwares neutros, mas que são comumente usados com propósitos maliciosos, principalmente voltados para espionagem e roubo de informação. Para conter as novas investidas deste tipo de perigo, é necessário ter programas que sejam específicos ou que tenham embutido em seu código a lógica necessária para fazer a detecção deste tipo de programa. Porém, não adianta somente este software ser capaz de detectar, ele também deve mostrar isso ao usuário de uma maneira que seja compreensível e capaz de se entender o que está acontecendo com o seu computador e o que está errado. Deste modo, este trabalho apresenta uma avaliação de softwares voltados para sistemas Linux, que fazem detecção de *Rootkits*, e obteve-se resultados bem interessantes em relação aos programas disponíveis, embora nem todos tenham sido positivos e/ou satisfatórios.

Palavras-chave: Rootkits, Malwares, Linux, Unix, Programas de Detecção, Detecção.

Abstract

The computational world has changed, technologies have developed in software and hardware, but as new benign applications have appeared and others have been improved, so have malignant applications. Rootkits should be neutral software, but they are commonly used for malicious purposes, mainly aimed at spying and stealing information. In order to contain the new attacks of this type of danger, it is necessary to have programs that are specific or that have embedded in their code the logic necessary to detect this type of program. However, it is not only useful for this software to be able to detect, it should also show this to the user in a way that is understandable and capable of understanding what is happening to your computer and what is wrong. Thus, this work presents an evaluation of software aimed at Linux systems, which detect Rootkits, and obtained very interesting results in relation to the available programs, although not all were positive and / or satisfactory. **Keywords:** Rootkits, Malware, Linux, Unix, Detection, Detection

Programs.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Acrônimos

AIDE	Advanced Intrusion Detection Environment
AIS	Artificial Immune System
AU	Advanced User
BU	Beginner User
CPU	Unidade Central de Processamento
DMA	Direct Memory Access
DINF	Departamento de Informática
EU	Expert User
GB	GigaByte(s)
IE	Intermediate User
IS	Sistema Imunológico
LKM	Loadable Kernel Module
NU	Nearby User
OP	Open Source
PPGINF	Programa de Pós-Graduação em Informática
RAM	Random Access Memory
SO	Sistema Operacional
TEHN	The Ethical Hacker Network
UFPR	Universidade Federal do Paraná
VMs	Máquinas Virtuais (Virtual Machines)

Capítulo 1

Introdução

A todo momento surgem novos softwares ou atualizações para softwares mais antigos, fazendo correções de funcionalidades ou melhorando-as. Porém, quando se trata de programas que visam melhorar a segurança dos Sistemas Operacionais, as ferramentas para Linux não são tão comuns quando as de sistemas Windows, isto porque não é tão habitual aparecer ameaças para este tipo de SO.

Uma ameaça antiga, mas que tem voltado a atuar novamente, são os *rootkits*, programas que podem causar modificações em várias funcionalidades no sistema, ou simular o sistema como um todo, cujo propósito é, principalmente, se esconder dentro dos Sistemas Operacionais e interceptar informações que interessem ao atacante. É necessário que existam programas capazes de identificar esta ameaça e informar a vítima que o seu computador esta comprometido, de uma forma clara.

Existem, em pouca quantidade, programas para Linux que são usados para verificar o sistema em busca desses perigos, mas não se sabe qual deles é o mais indicado para um determinado tipo de usuário, como estes softwares abordam o problema, como apresentam seus resultados, se é possível que eles verifiquem máquinas externas, se o material disponibilizado pelos desenvolvedores auxilia verdadeiramente no momento de utilizar o programa.

Pela solidez que o Linux apresenta, a maioria dos usuários acredita que não existem *malwares* destinados para este SO, e talvez por esta razão softwares destinados a este fim sejam motivo de curiosidade em quesitos de forma de resposta e funcionamento. Entretanto, espera-se chegar a uma conclusão que seja capaz de informar aos usuários se o que existe hoje é capaz ou não de enfrentar este crescente perigo computacional.

Este trabalho tem como objetivo fazem a avaliação de programas para Linux, que façam a detecção de *rootkits*, demostrando a capacidade e colocando à prova as heurísticas e algoritmos que embasam as verificações dos programas de detecção escolhidos para produzir esta análise avaliativa. Assim como também deseja-se verificar se os conteúdos produzidos em caráter informativo, tanto de resultados quanto de ajuda, realmente conseguem fazer com que o usuário como utilizar o software de detecção de forma correta.

É necessário que seja feita uma avaliação dos programas voltados para análise de *rootkits*, principalmente porque deve ser chamada a atenção novamente para este tipo de *malware*, que mais uma vez está se tornado comum e ainda mais ameaçador do que no passado. Não se trata somente de saber informar, mas se o que temos disponível realmente é capaz de responder a este inimigo.

Esta pesquisa será dividida em 2 (dois) módulos. O primeiro, fará uma avaliação de material web e local (manuais e *mans*), além da interface apresentada (linha de comando ou interface gráfica), e para quais tipos de usuário o programa é mais indicado. O segundo módulo

fará uma avaliação dos resultados obtidos, o modo como estes resultados são mostrados pelos softwares (se são passiveis ou não de interpretação), o tempo de execução e quantas amostras de *rootkits* eles foram capazes de detectar.

Este trabalho esta dividido como segue. Capítulo 2, traz informações sobre *rootkits*, um pequeno histórico, o que são, quais os tipos, casos relatados, como se instalam dentro da máquina da vítima, como podem ser utilizados e como preveni-los. O Capítulo 3 informa sobre técnicas de detecção que podem ser usadas, outras práticas que podem ser adotadas, programas conhecidos, um paralelo entre softwares de análise e *rootkits*. No Capítulo 4, apresenta-se as amostras de *rootkits* que foram encontradas, como foram testadas e quais foram escolhidas. No Capítulo 5, apresenta os programas de detecção, de que forma foram escolhidos e uma avaliação preliminar deles. No Capítulo 6, tem-se as informações de onde e como foram realizados os experimentos e a segunda parte das avaliações dos programas. O Capítulo 7, traz a conclusão e possibilidades de trabalhos futuros.

Capítulo 2

Rootkits

2.1 Introdução

Neste capítulo será comentado sobre *rootkits*, que são um tipo específico de software e/ou também pode ser colocado como um conceito por definir/atribuir características singulares a outros tipos de programas, que inicialmente não as possuíam.

2.2 Definição e Características

A terminologia *rootkit* provêm da junção de duas palavras: *"root"*, referência ao tipo de usuário que possui todos os privilégios do sistema, e *"kit"* [TEHN, 2005], que define um conjunto de ferramentas programadas agregadas, que dispõem de várias funcionalidades. Ou seja, um *rootkit* nada mais é do que um conjunto de programas que permite uma presença permanente, indetectável em um computador [Greg Hoglund, 2005].

Este tipo de software tem como principal característica sua capacidade de se esconder dentro do sistema, ocultando arquivos e pastas. Podem também dar acesso remoto ao atacante, explorando, para isso, falhas em programas desatualizados ou que apresentem uma códificação não segura; fazer espionagem do sistema, para fazer ataques como *man-in-the-middle* e similares [Greg Hoglund, 2005].

Alguns tipos *rootkits* empregam uma técnica chamada *hook* ou *hooking*, que são funções manipuladoras ou ganchos que modificam o fluxo de controle. Ou seja, quando o *rootkit* executa, um novo *hook* é registrado no lugar de uma função veridica do sistema, quando a função é chamada, o *hook* será executado em seu lugar, mas ele chamará a função original em algum momento para manter o comportamento original perante ao sistema [Kong, 2007]. O comportamento está demonstrado na Figura 2.1.

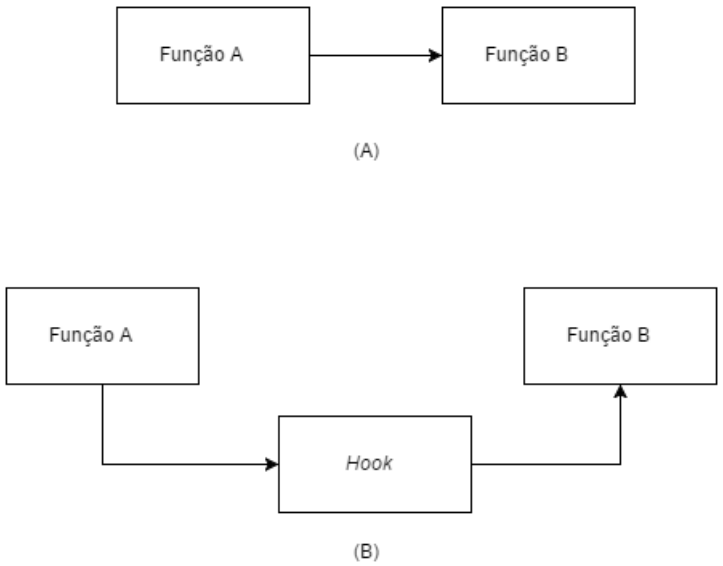

Figura 2.1: (A) Execução normal das funções, (B) Execução com *hook* [Kong, 2007]

2.3 Fatos Históricos

O *rootkit* mais antigo que se tem conhecimento foi feito para o sistema operacional da SUN, o SUN OS, em 1990 por Lane Davis e Steve Dake [Wikipedia, 2016b]. As versões iniciais desse tipo de software, cuja funcionalidade não fora divulgada, eram tão simplórias, que foram facilmente detectadas pelo programa Tripwire [Wikipedia, 2016a], produzido na época pela empresa Tripwire, Inc., nos anos 2000.

Para os sistemas Unix, os primeiros *rootkits* começaram a aparecer no início dos anos 90, mas não tiveram um desenvolvimento notado ou não obtiveram um que causasse a demonstração de algo notável.

Para os sistemas operacionais distribuídos pela Microsoft, o primeiro *rootkit* encontrado foi no SO do Windows NT

Em 1995, um programador chamado Jeffrey Ritcher demonstrou, com linhas de código em seu livro *"Advanced Windows"*, várias técnicas para interceptar *system calls* em uma versão do Windows em modo usuários, que levaram a uma fasta produção de *rootkits* que se inspiravam ou as utilizavam diretamente [Shevchenko, 2008]. Criado em 1999 por Greg Hoglung, que lhe deu o nome de *"NT Rootkit"*, este software não era unicamente um *rootkit*, mas também possuía características de *trojans*, tornando-o uma das primeiras versões maliciosas e focada em roubo e informações e outras operações ilegais [Wikipedia, 2016b].

Já em 2001, os *rootkits* começaram a ganhar certa fama, sendo um dos temas de artigos da revista Phrack, que descrevia como um software poderia modificar partes do *Kernel*, isentando qualquer possibilidade de detecção, pois nesta época havia bastante dificuldade para um software conseguir chegar a esta camada, por todos os artifícios de segurança [Oliveira, 2008].

Outro caso envolvendo *rootkit* ocorreu entre 2004 e 2005, na Grécia [Leyden, 2007], um incidente que foi marcado como um dos mais notórios escândalos de escutas telefônicas. Cerca de 100 aparelhos da Ericson AXE, pertencentes aos funcionários do alto escalão do governo grego, outras autoridades e profissionais de renome foram afetados por esta ação ilegal. Por uma falha do aparelho, um *spyware* modificado desviou conversas telefônicas de assinantes da empresa Vodafone, permitindo que as chamadas fossem monitoradas e, provavelmente, gravadas

por criminosos. Acredita-se que as mensagens de texto também tenham sido alvo, tendo sua rota de destino alterada.

Ainda em 2005 também ocorreu um acontecimento envolvendo a empresa Sony. A companhia vendeu CDs com um software embutido, que só se fazia presente e se instalava quando a mídia era colocada dentro do leitor de CD do computador. A ideia era impedir que fossem feitas cópias piratas [Wikipedia, 2016b].

Mantendo-se no ano de 2005 [Shevchenko, 2008], cerca de 80% dos *rootktis*, em circulação, eram algum tipo de variação do HacDef[1] e do Haxdoor[2]. Os primeiros *Trojans* que ganharam funcionalidades de *rootkit* foram o *Rbot* e *Sdbot*, pois os programadores maliciosos começaram a ver vantagens e obtenção de ganhos financeiros se conseguissem fazer a adição de certos recursos em seus softwares, o que resultou na integração das características *rootkit* dentro deles.

Já em 2006 [Shevchenko, 2008], *Worms* também já tinham recebido as funcionaçidades de *rootkit*, cujos mais conhecidos são alguns de email como *Bagle*, *Goldun*[3], *Rustock*[4]. Outra vitima de escândalo foi a própria *Kaspersky* Antivírus, cujo software escondia alguns arquivos do sistema dos usuários, por se tratar de uma época conturbada e o aparecimento de muitos softwares com esse tipo de atribuição, iniciou-se discussões sobre o uso deste tipo de abordagem e muitas pessoas acabaram banalizando sua utilização.

Ainda em 2006 [Shevchenko, 2008], no meio acadêmico, os *rootkits* também começaram a se popularizar. *SubVirt*[5] [King et al., 2006], *BluePill*[6] [Rutkowska e Tereshkin, 2006] e *Vitriol*[7] [Zovi, 2006]. Todos os três atuam em uma camada definida como denominada hipervisor, responsável pelo gerenciamento de máquina virtuais.

Em 2005 e 2007 [Shevchenko, 2008], outras duas provas de conceito foram lançadas, mas estas visavam o setor de boot, são eles respectivamente *eEye Bootroot*[8] [Soeder e Permeh, 2005] e *Vbootkit*[9] [Kumar e Kumar, 2007].

Em 2009, foi encontrado o primeiro *rootkit* rodando no sistema MAC OS X, acredita-se que a falha para a sua entrada, no aparelho da Apple, tenha sido alguma lacuna deixada nos códigos do Kernel do MacOS.

Depois de 2009 a meados de 2014, quase nenhum caso foi relatado envolvendo *rootkits*, embora acredite-se que seu desenvolvimento não tenha parado. Conforme os telefones foram se modernizando, alguns deles foram voltados para o sistema Android e mais versões para o sistema MacOS foram liberadas, mas nenhum caso preocupante foi relatado. Em contrapartida, no âmbito acadêmico continuou-se os estudos e provas de conceitos, chegando ao nível de placa. *Rootkits* instalados nos firmwares e no hardware das placas do computador, como o *JellyFish*, desenvolvido por investigadores [Simioni, 2015].

Em 2014 foi relatado um novo software com traços de *rootkit*, voltado para invasão de sistemas Unix. Apelidado de *XOR.DDoS* [Kálnai, 2015], forma uma botnet para ataques distribuídos de negação de serviços. Em 2015, foi relatado o aparecimento do *Umbreon*

[1] Seu nome é completo é na verdade *Hacker Defender*, feito para sistema Windows em 2002, embora seja somente uma ferramenta e não um *rootkit* de fato, ele podia esconder arquivos, processos e chaves de registros, e funcionava em modo usuário.

[2] É um *backdoor* que ocultava a sua presença no sistema, funcionava em modo *kernel*.

[3] É um Trojan-Spy.

[4] É um Mailbot.

[5] Cujos autores são Samuel T. King, Peter M. Chen, Yi-Min Wang, Helen Wang, Jacob R. Lorch, Jay Lorch.

[6] Cujos autores são Joanna Rutkowska e Alexander Tereshkin.

[7] Seu autor é Dino A. Dai Zoni.

[8] Cujos autores são Derek Soeder e Ryan Permeh.

[9] Cujos autores são Nitin Kumar e Vipin Kumar.

[TrendMicro, 2016a], especula-se que seu desenvolvimento tenha começado em meados de 2011, mas foi somente a partir do ano passado que ele ficou conhecido. Por atacar em nível de usuário e usar uma técnicas de bibliotecas dinâmicas, o *rootkit* nomeado como o Pokémon das Sombras, tornou-se uma das maiores ameaças por sua disseminação rápida e ausência de plataforma (por seu código quase completamente sendo escrito puramente em C, tendo somente algumas ferramentas escritas em Python e Scripts de Bash), uma vez que ele pode agir tanto em x86, x86-64 e ARM[10]. Segundo especulações seu maior alvo são Linux Servers.

2.4 Tipos de Rootkits conhecidos

Atualmente, os *rootkits* são dotados de inúmeras capacidades e características. Porém isto torna cada vez mais difícil e trabalhoso encontrar um modo de classificá-los de uma forma padrão.

Cada vez mais, seus códigos-fontes passam por alterações para conseguirem burlar as proteções de antivírus e *firewalls*. Como são incumbidos de novas tarefas, a complexidade de seu código aumenta, e não é mais passível tentar rotulá-los por suas capacidades e/ou habilidades. Também ocorre, muitas vezes, ter muita falta de informação ou excesso dela, e isso impossibilita que seja feito um agrupamento único para esse tipo de software.

Portanto, será abordada uma categorização que utiliza o exemplo de divisão de camadas do computador, por ser a única forma restante de diferenciação real entre os *rootkits*, pois cada *rootkit* tem um alvo específico dentro de computador. Essa divisão segue o padrão mostrado na Figura 2.2.

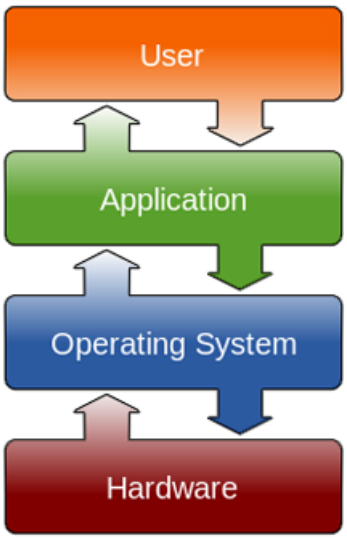

Figura 2.2: Divisão de camadas do computador

Usando a Figura 2.2, como apoio, serão mostrados 4 (quatro) tipos de rootkits:

Nível de Usuário: As camadas de aplicação e de usuário, como mostrado na Figura 2.3, se remetem a todos os programas que são executados pelo usúario (inclusive do usuário-root) e softwares que não precisam de acessos privilegiados para executarem suas funções.

[10]RaspberryPi.

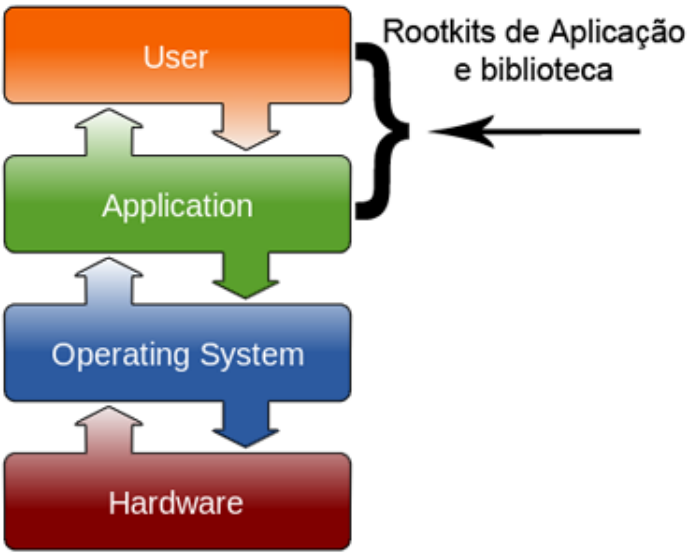

Figura 2.3: Onde se localizam os rootkits do nível de usuário

Os *rootkits* dessa camada, são os mais convencionais e os mais fáceis de serem identificados. Eles tendem a se aproveitar de certas aplicações que rodam neste nível para se instalar, modificar comandos e bibliotecas para ocultar a sua presença. Dois exemplos de *rootkit* dessa camada são:

- De comandos [Manap, 2001]: Ele altera alguns comandos como *ls*, *find*, *ps*, *top*, *netstat*, com a intenção de ocultar sua localização, o que está fazendo, o que está enviando, se está ativo e o que está monitorando.

- Da *libc*: São feitas implementações alternativas de suas funções, para burlar autenticações e modificar seus comportamentos, com o intuito de ocultar a sua presença dentro do sistema.

Destes dois exemplos, o segundo é mais prejudicial, pois é capaz de alterar códigos usados muito frequentemente pelas aplicações do usuário e/ou pelo próprio Sistema Operacional. Existe a possibilidade de essa alteração causar *bugs* desconhecidos no sistema, ou levá-lo a um estado de processamento indeterminado, onde não se sabe como ele irá se comportar, se irá agir da forma esperada e até que ponto continuará executando conforme uma falsa normalidade. Um *rootkit*, desta camada, que ganhou notoriedade recentemente é o *Umbreon* [TrendMicro, 2016a], seu nome foi dado em homenagem ao pokémon que se esconde nas sombras, que também é uma das principais características deste tipo de software. Ele atual em nível de usuário e se utiliza da oportunidade de criação de bibliotecas dinâmicas para conseguir dados e informações, existe a especulação que seu principal objetivo são Linux Servers, mas ele consegue infectar até mesmo softwares embarcados.

Nível de *Kernel*: Esta camada é constituída pelo programa que é a parte central do Sistema Operacional, ele tem o controle total sobre tudo o que acontece dentro do SO (sua própria camada) e das camadas acima (usuário e aplicação). São os mais comuns nos sistemas

Posix. O primeiro *rootkit*, que surgiu para este SO, foi nesta camada e usava as mesmas técnicas utilizadas para invadir sistemas na década de 80. [TEHN, 2005].

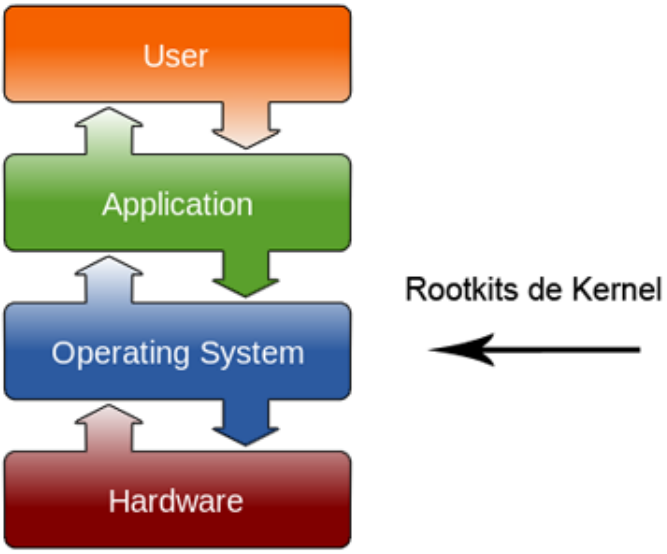

Figura 2.4: Onde se localizam os rootkits do nível de Kernel

Eles se escondem dentro do *Kernel*, como mostra a Figura 2.4. São de difícil detecção, pois possuem os mesmos privilégios que o próprio Sistema. Entre suas capacidades, estão alterar *systens calls*, incluindo aquelas que listam os módulos de *kernel* instalados, podem adicionar ou substituir partes do código do *kernel*, incluindo drivers e dispositivos agregados. Podem ser de grande impacto no sistema, uma vez que suas alterações podem causar instabilidade no sistema, pelos códigos inseridos terem a possibilidade de conter *bugs*. Um exemplo de *rootkits* dessa camada é o *Bootkit* [Wikipedia, 2016b], que substitui o boot loader do sistema por um sob o controle do atacante. Quando o sistema é iniciado ou reiniciado, a BIOS carrega este código novo e o SO estará sob o domínio do invasor. Alguns exemplos de *bootkit* conhecidos são o *SucKit* [Wikipedia, 2016b], para sistema Unix, o *Stoned Bootkit*[Kleissner, 2007] para Windows, *eEye Bootroot*[Soeder e Permeh, 2005] e *Vbootkit*[Kumar e Kumar, 2007].

Nível de Hipervisor: Um hipervisor é uma camada de software que realiza a virtualização do hardware, permitindo a construção de máquinas virtuais. *Xen, VMWare, KVM e VirtualBox* são exemplos bem conhecidos de hipervisores. Um rootkit de hipervisor faz uso das capacidades de virtualização para se instalar entre o nucleo do sistema operacional e o hardware da maquina, como indica a a Figura 2.5.

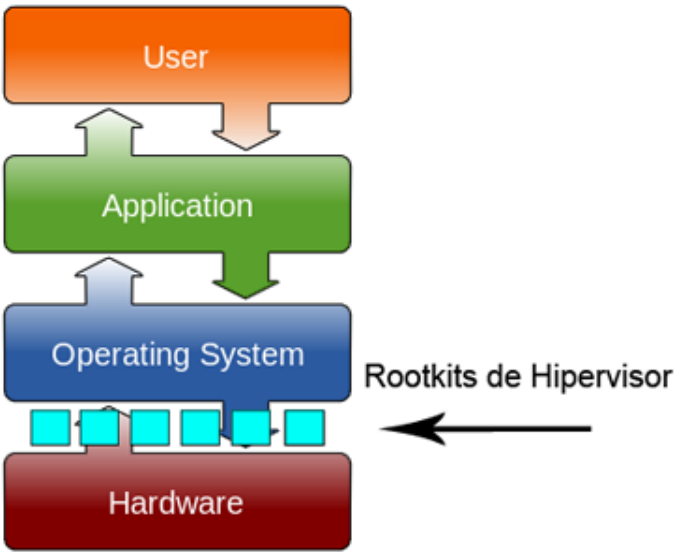

Figura 2.5: Onde se localizam os rootkits do nível de Hipervisor

Os *rootkits* capazes de rodar neste nível são praticamente invisíveis, pois não deixam qualquer traço de modificação de códigos, ou seja, não utilizam hooks[11]. Eles fazem o SO acreditar que está executando sobre um hardware real, enquanto está, na verdade, executando dentro de uma maquina virtual controlada pelo atacante. Eles funcionam de forma muito similar a um programa de máquina virtual, mas de um modo imperceptível. Sua detecção é extremamente complicada, por não deixa pistas de sua presença, além de não necessitar que o sistema operacional seja reiniciado para concluir a sua instalação. Dois exemplos de *rootkit* deste tipo são o famoso *Blue Pill* [Rutkowska e Tereshkin, 2006], criado pelos pesquisados Joanna Rutkowska e Alexander Tereshkin; o *SubVirt* [King et al., 2006], desenvolvido por pesquisados da Microsoft em conjunto com a Universidade de Michigan; e o *Vitriol*[Zovi, 2006]. Todos esses *rootkits* são provas de conceitos acadêmicos, ainda não foram encontrados softwares deste tipo fora de um ambiente controlado, mas não se pode refutar a sua existência

Nível de Firmware/Hardware: É a camada mais baixo nível de todo o computador, localiza-se na área das placas ou componentes físicos que estão dentro do computador (hardware) e os programas que rodam nas placas (firmware).

[11]É a capacidade que o *rootkit* possui de alterar ou injetar novas linhas ou trechos de código em partes sensíveis do sistema, como a *Libc* ou a tabela de *System Calls*.

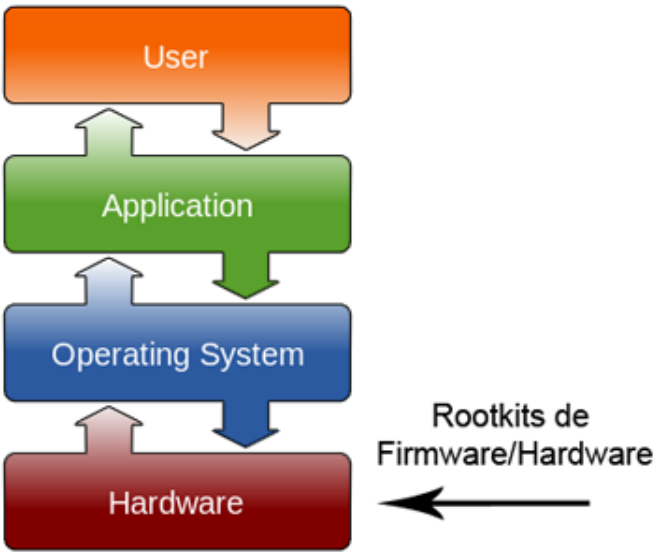

Figura 2.6: Onde se localizam os rootkits do nível de Firmware/Hardware

Como mostrado na Figura 2.6, os *rootkits* dessa camada tem como alvo, principalmente, os firmwares. O que eles fazem é criar uma nova imagem destes programas para ser executada em cima do hardware. É um local extremamente estratégico, pois nenhum programa de verificação consegue chegar a um nível tão baixo. Um firmware bastante visado é do da BIOS, como por exemplo da *Mother Board*, pois todas as outras placas estão conectadas nela, todas as informações do que o usuário está fazendo passam por ela. Um exemplo de uso deste tipo de *rootkit*, foi a adulteração dos leitores de cartões de crédito por criminosos europeus descrita no capitulo 9 de [Simpson et al., 2012]. O *malware* funcionava da sequinte maneira: quando um cartão era lido, o dispositivo de leitura era interceptado e transmitia os detalhes deste cartão de crédito através de uma rede móvel. Para este crime foi usado um *rootkit* de firmware com alteração na BIOS.

Existem outras abordagens, embora sejam mais científicas, como a de John Heasman [Rohr, 2006a], em que ele descreve um modo para que rootkits se escondam dentro das placas de vídeo e de rede, prevendo que eles persistam mesmo com a formatação e reinstalação do sistema operacional. Outra ocorrência é o *Jellyfish* [Simioni, 2015], criado por investigadores para propósitos acadêmicos, cujo código está disponível no Github. Este *rootkit* utiliza OpenCL, tecnologia usada em placas da Nvidia e AMD, e acessa as informações via DMA. Sua proposta é que ele seja realmente indetectável graças a essa abordagem, pois ele teria acesso direto aos dados.

2.5 Instalação

Existem muitos modos de um *rootkit* se instalar no sistema operacional. Em várias ocorrências, o próprio sistema contém alguma falha, podendo ser de código de algum programa que está desatualizado e por isso pode ser usado como porta de entrada (softwares como Java e Flash são comumente usados neste tipo de abordagem por seus códigos desatualizados

apresentarem buracos que podem facilitar a invasão), ou *Phishing*[12], algum site que foi infectado e está sendo usado como meio de disseminação de *malwares*.

Através da rede de internet, via acesso a sites ou por vulnerabilidades de outros programas que estão rodando no sistema, forma-se uma ponte entre o computador da vítima e do atacante, como mostra a Figura 2.7 [Manap, 2001]. Geralmente, a escolha dos alvos é aleatória, qualquer um que esta na rede pode ser um alvo em potencial, mas existem abordagens que visam vítimas específicas como usuários de algum banco ou até mesmo algum trabalhador ou pessoas ligadas a uma determinada empresa ou órgão, como foi o caso das escutas telefônicas envolvendo a Vodafone.

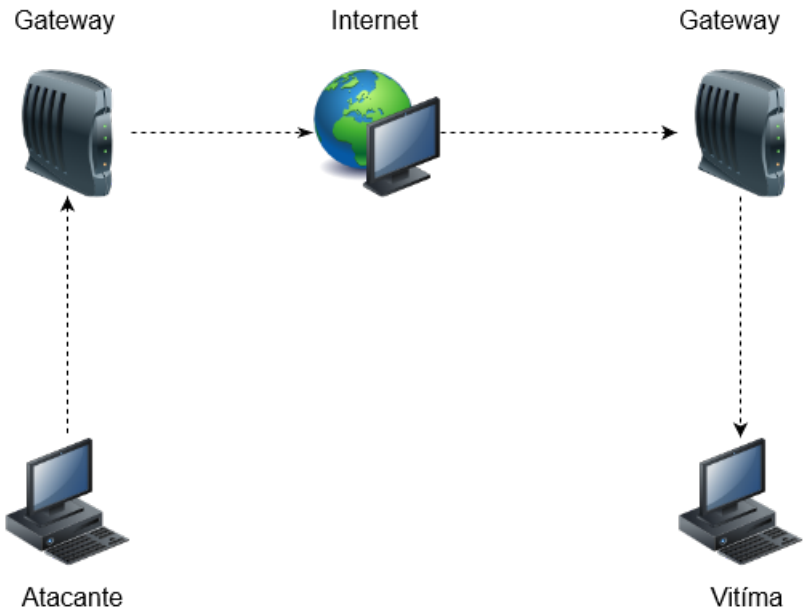

Figura 2.7: Ponte entre o computador do atacante e o computador da vitima. Foi estabelecida uma conexão entre eles.

O software vem pela rede e se infiltra dentro do sistema do usuário. Algumas vezes é necessário que ocorra uma reinicialização do sistema para que o *rootkit* se instale (alguns de *kernel*), em outros casos não (alguns que ficam na camada de aplicação e na de usuário). Na Figura 2.8, o *rootkit* já está dentro da máquina da vítima, preparando-se para se instalar em uma de suas camadas.

[12]São ataques de engenharia social, que levam o usuário a executar e/ou instalar softwares que não pretendia ou não tem conhecimento real de sua funcionalidade.

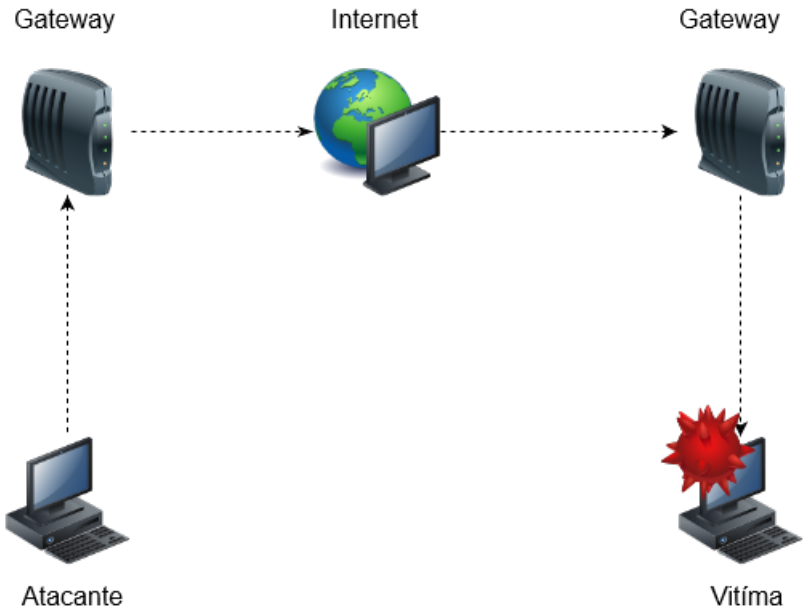

Figura 2.8: O rootkit já está dentro da máquina atacada.

Na Figura 2.9 é mostrada a instalação do *rootkit* em uma das camadas do computador. Uma vez que consiga se instalar com sucesso, ele irá executar modificações no sistema para se esconder dos programas de segurança instalados ou da detecção de outros softwares que podem vir a ser instalados posteriormente. As modificações sofrem variações pelo tipo de *rootkit*. Os que ficam na camada de aplicação e de usuário tendem a modificar poucos registros, já os de *Kernel* tendem a alterar mais e acabam por controlar várias partes do SO. Os de hipervisor controlam a interface entre o SO e o hardware, enquanto os de firmware e hardware controlam o software do componente em que atacaram.

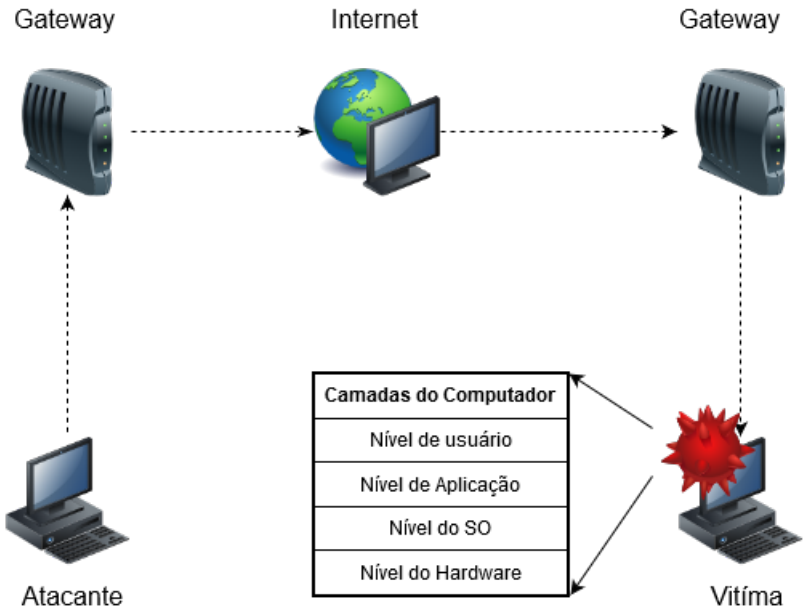

Figura 2.9: O rootkit é instalado na maquina da vítima

Com a instalação executada com sucesso, o atacante pode ter pleno acesso a informações e dados do computador invadido. Um detalhe a ser lembrado, é que quanto mais baixo o nível,

mais complicada é a detecção do software invasor, mas em contrapartida, mais complexo será o seu desenvolvimento, como por exemplo, se um *rootkit* se infiltra na placa de rede, ele deve saber como montar e manejar as mensagens. Então, a complexidade do programa aumenta demasiadamente, por conta da parte responsável pela interpretação dos dados.

Na Figura 2.10, como a vítima está conectada na rede, seu computador pode servir de ponte para que atacante consiga alcançar e infecte outras máquinas.

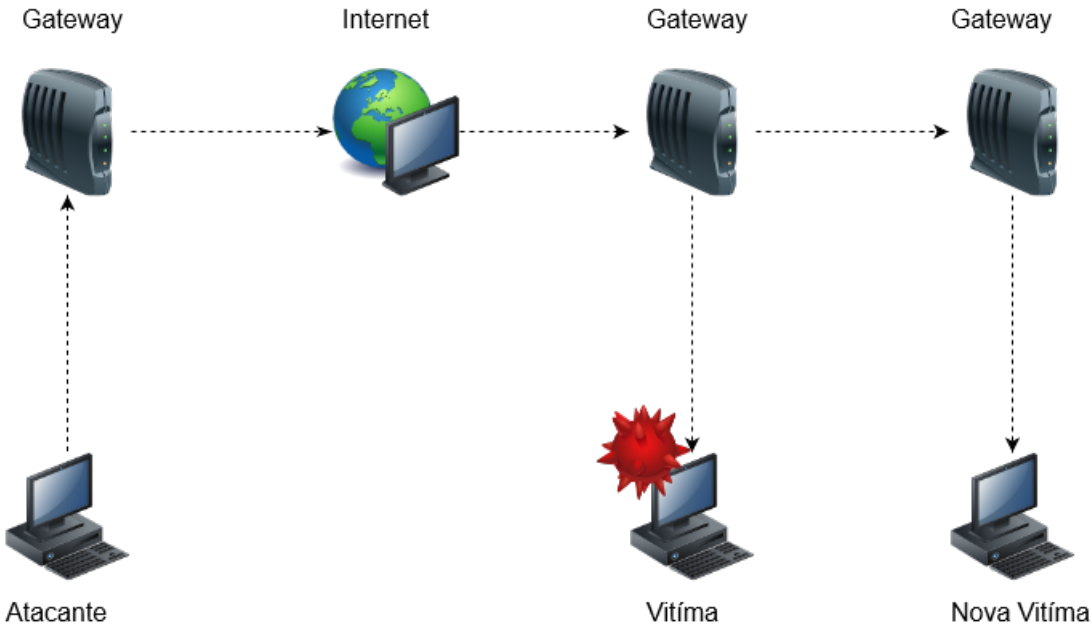

Figura 2.10: O computador da vitima passa a ser monitorado pelo atacante e pode "infectar" outros computadores.

2.6 Utilização

Embora os *rootkits*, em sua maioria, sejam usados como malwares, existem meios para que ele seja usado de forma legal e, até mesmo, benéfica. Alguns usos seriam: (1) Um monitoramento eficiente dos dados acessados/guardados/alterados por funcionários em empresas de quaisquer ramos e que tenham livre uso de ferramentas como computadores, tablets e notebooks; (2) Auxiliar na obtenção de provas contra criminosos, desde que isso seja legalizado pela justiça, neste caso o *rootkit* funcionaria igual a um grampo telefônico; (3) Proteção/ verificação e possível manutenção dos firmwares de placas, onde o próprio sistema teria autonomia para dar *feedbacks* ao usuário e ou o fabricante sobre a usabilidade daquele determinado componente, se houve algum tipo de alteração e porque.

Outra prática não lícita envolvendo *rootkits*, mas no meio de jogos online, onde jogadores usam desse tipo de software para esconder o uso de trapaças (os famosos *cheats*) [Rohr, 2005]. Os *gamers* aproveitaram-se das capacidades de ocultamento que o *DRM/rootkit* [13] pode promover para esconder suas trapaças dos softwares que fazem a detecção deste tipo de abordagem, assim impendindo que o *bot de cheating* seja descoberto.

[13]Este *rootkit* é o mesmo usado pela Sony, anos atrás para impedir que usuários fizessem cópias piratas dos CD's de aúdio.

2.7 Prevenção

Rootkits estão em um nível de ameaça tão grande, que desafiam até mesmo muitos programas de detecção, remoção e análise. Não se sabe com certeza se todos os seus arquivos são excluídos, ou se algum resquício ficou para que possa restaurá-lo posteriormente.

A única forma eficiente de deixar o computador livre é prevenindo a infecção. Algumas práticas simples podem ser usadas: (1) criar uma conta secundária, com privilégios limitados[14]; (2) Manter o sistema sempre atualizado, isso impede que sejam abertas brechas no sistema; (3) Sites Maliciosos ou de conteúdo duvidoso devem ser evitados, pois sites que não se conheça a procedência, assim como baixar algum conteúdo dos mesmos podem resultar em arquivos indesejados; (4) Anexos de emails, deve-se suspeitar de anexos relacionados a serviços que não foram solicitados ou que não sabe da procedência, e pessoas com as quais nunca teve ou tem pouco contato; (5) Evitar redes P2P, são redes muito usadas para a distribuição de conteúdo pirata e por isso estão cheia de bots, e muitos deles podem conter *rootkits* dentro de si; (6) Evitar ou não fazer uso de *cracks*.

Seguindo estes passos simples, existe uma grande chance de se evitar que o computador seja infectado por softwares dessa natureza, pois são extremamente complicados de serem retirados. Então, a prevenção acaba sendo o melhor caminho.

2.8 Conclusão

Rootkits são softwares, que como tudo no mundo da informática, sofreram modificações e *updates* com o passar dos anos. Embora existam poucos[15] que são verdadeiramente criados, a grande maioria sofreu algum tipo de modificação para ter as mesmas características e, em um balanço geral, são softwares escritos para fins maliciosos.

Embora seu poder destrutivo tenha dado lhe dado notoriedade, mesmo que que negativa, depois de 2009, no meio comercial não encontra-se nenhum grande relato de casos envolvendo *rootkits*. Existem duas suposições para este fato: (1) Os casos não foram reportados e (2) Algumas atualizações feitas nos sistemas e/ou nas placas de hardware para deixar o computador menos vulnerável a este tipo de infecção, porém, não se sabe a razão real, embora no âmbito acadêmico eles não tenham deixado de ser foco de constantes estudos.

No meio computacional hoje, existem tantas variações de ameaças com as habilidades de *rootkit*, que é difícil de classificá-las e enquadrá-las e também de saber o que é realmente um *rootkit* e o que é um vírus. Embora, todos considerem que eles [*rootkits*] sejam parte do mundo dos softwares maliciosos, seja por suas características, seja por seu modo de instalação, na verdade, são somente softwares que podem ser usados para atribuições positivas, basta que seus códigos sejam usados de forma correta. Todo e qualquer software esta passível de ser usado por pessoas com intuitos maliciosos, e com os *rootkits* não é diferente. Existem várias áreas nas quais poderiam ser empregados para ajudar não só peritos como também donos de empresas. Porém, é igualmente difícil contornar toda a visão de ameaça para dar uma chance ao grande potencial de ajuda que estes softwares têm.

Portanto, é uma tarefa delicada quando se trata sobre *rootkits* em todos os aspectos, mas talvez, para enfrentar as ameaças que nascem na *Deep Web* e até mesmo da cabeça dos pesquisadores, seja o momento de estudar e aplicar estes softwares em algo positivo para a sociedade.

[14] O Linux faz isso por padrão, mas no Windows deve ser configurado manualmente.

[15] Poucos, no modo de dizer, para representar alguns milhares, que hoje no mundo das ameaças é algo irrisório.

Capítulo 3

Detecção e Remoção de Rootkits

3.1 Introdução

O que se espera quando se presencia um ataque de *malware*, no geral, existem algumas características que podem ser percebidas no dia-a-dia, no uso do aparelho. Abas nos navegadores de internet começam a se abrir sozinhas mostrando anúncios indesejados, o sistema fica lento, pode ocorrer uma falha em algum processo com alguns poucos minutos de uso que impossibilitam o uso, o sistema não inicia, entre outras centenas de milhares de sintomas que podem ocorrer.

Quando se trata de *rootkits*, dificilmente existe a ocorrência de algum sintoma, pois sua principal característica é a de tentar ocultar a sua presença, tanto do usuário, quanto do próprio SO e, especialmente, dos softwares e mecanismos de defesa.

Mesmo com certas dificuldades e a grande capacidade que os *rootkits* têm de se esconder, existem meios de detectar sua presença dentro dos sistemas. Podem ser usados programas, mecanismos e técnicas especializados para encontrá-los.

3.2 Técnicas de detecção

Por causa dessas características tão singulares os *rootkits* são dificilmente detectados. Hoje, devido à sua importância e grande preocupação com o roubo de informações, existem vários tipos de softwares produzidos para pesquisar o Sistema Operacional, a fim de encontrar algum rastro ou, com sorte, o próprio *rootkit*. Alguns tipos de análises [Bunten, 2004] [Butler et al., 2006] usadas por esses programas são:

Detecção heurística ou comportamental: É uma abordagem baseada em comportamento, e ela se dá através da inferência da existência de um *rootkit* dentro do sistema da vítima, mediante algumas análises iniciais, como modificação de arquivos, frequência de chamadas de alguma determinada API ou *System Call*, uso da CPU, ou até mesmo logs de um analisador de pacotes, antivírus, *firewall*, podem apresentar indícios da presença de um software invasor no sistema. É um modo de análise bem complexo que requer bastante conhecimento em sua utilização, devido ao grande número de falsos positivos que este tipo de verificação acarreta.

Detecção baseada em integridade: Verificadores de integridade mantêm um banco de dados das *fingerprints* dos arquivos e, com isso, analisam se estes [arquivos] foram modificados ou não. Está verificação pode ser feita de 3 (três) maneiras: (1) em uma varredura no sistema; (2) por acesso a este arquivo em questão; (3) também podem ser verificados enquanto

são carregados. A verificação por *fingerprints* é mais segura do que usar *checksum* ou *hash*, pois evita-se que aconteça um mesmo *checksum* ou *hash* para arquivos diferentes. Apesar de ser um método mais seguro, é importante que a base de dados seja protegida de possíveis alterações não autorizadas [Swimmer, 2008].

Detecção baseada em assinatura: É o mesmo tipo de detecção usada pela grande maioria dos antivírus e anti-spywares. Também conhecida como Identificação por Impressão Digital. O programa guarda dentro de seu banco de dados um pedaço do código do *malware*/vírus/software malicioso e vasculha o sistema em busca desses programas cujo pedaço de código coincida com o que ele tem salvo em seus dados. É um sistema bem preciso, mas ruim quando se trata de eficiência, pois deve ser atualizado de forma constante, para que as bases de dados não fiquem obsoletas em reação aos novos vírus que aparecem todos os dias. Também é um sistema um pouco falho quando se trata de rootkits que não são muito conhecidos, ou se seu código é mutável. Outro problema com este tipo de detecção é que cada base de dados classifica a "impressão digital" à sua maneira, não existindo um padrão. Então, um *trojan* pode ser confundido com um *rootkit*, que pode ser considerado um *spyware*, que no outro banco está como *worm*. Isso dificulta em saber o que realmente estava infectando o sistema.

Detecção baseada em diferença: Também conhecida como "Cruzamento de visão", esta forma de detecção faz uma comparação dos dados "limpos"com os dados "sujos" retornados de alguma alteração, código, API, chamada do sistema [Wikipedia, 2016b]. É uma abordagem simples, mas pode vir a não ser muito eficiente se o *rootkit* tiver capacidade para detectar que algo deste tipo foi instalado no sistema, para assim ajustar seu comportamento para que todas as mudanças causadas não sejam detectadas, passando completamente despercebido.

3.3 Outras Práticas

Uma prática extremamente comum entre os usuários é a de tentar executar os programas de detecção e remoção na máquina infectada, a fim de tentar descobrir e remover o que está causando problemas em seu computador. Porém, está tática não traz garantias de que o processo de retirada do *rootkit* foi eficaz, pois pode ser um falso negativo. Isso pode ser afirmado porque o sistema sofreu algum tipo de alteração devido ao *rootkit* instalado. Se for um que atinja camada de usuário, alterações nas bibliotecas podem ter sido feitas, o que comprometeria certas funcionalidades dos softwares de detecção que precisem dessas bibliotecas que foram alteradas. Se a camada do *Kernel* foi o alvo escolhido, o núcleo estará comprometido e todas as chamadas do sistema que o programa de detecção fizer poderão resultar em falsos negativos. Se o *rootkit* atingir a camada de hipervisor ou firmware/hardware, dificilmente ele irá ser detectado por alguma ferramenta, pois nenhuma delas faz verificação em um nível tão baixo, o que torna a descoberta da infecção e a remoção muito complicadas.

O mais comum, e indicado por pessoas da área de segurança, é fazer uma análise dos discos do sistema possivelmente infectado em outro computador, ou inicializá-lo a partir de um dispositivo externo, como um pendrive ou CDROM, mas sem fazer o *boot* a partir do disco interno da máquina sob suspeita, pois acarretaria os mesmo problemas já listados anteriormente de se executar os programas instalados no sistema infectado. Existem outras abordagens muito mais seguras, como um hardware que faz monitoração de barramento, mas esta abordagem está fora do escopo deste trabalho.

3.4 Programas para detecção

Atualmente existem vários programas tanto para sistemas Linux, quanto para sistemas Windows, que executam análise também para encontrar *rootkits*, alguns deles são:

- Linux:
  - AIDE [Lehti et al., 2016]
  - CHKRootkit [Murilo e Steding-Jessen, 2014]
  - Lynis [Audit, 2016]
  - RKHunter [Boelen, 2014]
  - Tripwire [Tripwire, 2016]
- Windows:
  - Bit Defender Rootkit Remover [Bitdefender, 2016]
  - F-Secure BlackLight [F-Secure, 2016]
  - Kaspersky [Kaspersky, 2016]
  - Malware Bytes Anti-Rootkit [Malwarebytes, 2016]
  - RootkitRevealer [Sysinternals, 2016]
  - TrendMicro Rootkirt Buster [TrendMicro, 2016b]
  - UnHackMe [Sokolov, 2016]

3.5 Programas de detecção vs rootkits

Programas de detecção e remoção, assim como antivírus, anti-spywares, *firewalls*, deveriam auxiliar e evitar que softwares maliciosos se infiltrem dentro do Sistema Operacional, mas já houve casos de grandes empresas, que são responsáveis por softwares de segurança, que utilizavam implementações semelhantes a *rootkits* em seus códigos [Rohr, 2006b], além de outros programas de detecção que já foram usados por rootkits, para camuflarem ainda mais sua presença [ARIS-LD, 2012].

Um caso envolvendo a Symantec, empresa responsável pelo Norton Antivírus, e outro envolvendo a Kasperky, onde ambas as empresas estavam usando códigos com camuflagem semelhante a *rootkits* para esconder certos processos do usuário. Embora as funções tenham sido documentadas, e estejam vinculadas a programas de segurança, é possível que algum *malware* pudesse tirar proveito dessas mesmas funções para ocultar o seu próprio *rootkit*, tornando-o praticamente indetectável pelo sistema.

Outras ocorrências que vem acontecendo desde meados de 2008, é a utilização de programas de detecção e remoção para deixar o computador de vítima desprotegido. Algumas ferramentas, destacando-se GMER, Partizan e Avenger [ARIS-LD, 2012], foram alteradas para remover softwares de segurança, como antivírus e plug-ins bancários, em vez de procurarem *rootkits* no sistema.

Não há notícias se os softwares para Linux também foram alvo de uma prática similar, todos os citados são direcionados ao SO Windows.

3.6 Conclusão

Cada vez mais os desenvolvedores de programas de detecção têm seus conhecimentos e habilidades colocados em xeque pelos desenvolvedores de tecnologia maliciosa. Embora existam grandes avanços nas análises, melhorando as técnicas ou tentando novos meios de implementação, em alguns aspectos parece que eles sempre estão um passo atrasados e muitas vezes não é somente pelos criminosos, mas também pelas mentes dentro das universidades.

Independente de sistema operacional e hardware, a capacidade de fazer *rootkit* para qualquer nível nas camadas é muito maior do que a capacidade e recursos dos softwares desenvolvidos para detecção e remoção desenvolvidos hoje. É como se fosse um aprendizado em cima do que está dando errado, ou como se esperasse que o problema acontecesse para então tentar resolver, uma estratégia com antecipação de movimentos quase não é utilizada, talvez porque seja difícil de prever qual parte do sistema será atacado, ou falhas que podem existir nos códigos que são ignoradas/passam despercebidas que podem ser usadas, ou existe uma dificuldade muito grande dos desenvolvedores de pensar como um atacante, em imaginar quais brechas seriam usadas, pois a quantidade de programas e sistemas que podem apresentar vulnerabilidades é muito grande.

Por fim, ficam inúmeras dúvidas sobre a eficácia real do que é usado nos softwares de detenção e remoção, e isso não só para *rootkit*, mas *malwares* no geral. Talvez seja o momento de se abordar outras técnicas, como técnicas bio-inspiradas, que são soluções baseadas na natureza. Um bom exemplo de metodologia, que pode ser utilizada, pode ser baseada no sistema Imunológico Humano. O IS é composto por várias células diferentes que atuam de formas especificas para proteger o corpo humano de invasores. Existem uma área de pesquisa que utilizam essa abordagem, ela é chamada de *Artificial Immune System* ou AIS, que são soluções computacionais baseadas no comportamento do IS. Já existem alguns mecanismos de detecção de spam baseados neste tipo de solução, talvez o próximo passo seria adaptá-los para sistemas de detecção de *rootkits*.

Capítulo 4

Amostras de Rootkits

4.1 Introdução

Neste capítulo serão apresentadas as amostras de *rootkits* que foram encontradas para sistema Unix. Primeiramente será explicada a motivação que acarretaram a escolha delas, como foram feito os testes, em seguida como elas foram instaladas nas máquinas, quais são essas amostras e outras informações que foram obtidas e quais serão usadas no experimento final.

4.2 Diretivas

4.2.1 Motivação

O primeiro grande desafio, quando se trata de desenvolver qualquer experimento ou avaliação envolvendo *rootkits* é encontrar amostras funcionais, a complicação aumenta quando se trata de sistemas Unix, pois as que são desenvolvidas, em grande maioria, são para invasão de servidores e roubo de informações e por isso não são distribuídas fora da *Deep Web*. Por esta razão optou-se por usar amostras desenvolvidas por usuários do GitHub, cujos códigos fossem compatíveis com a versão do *Kernel* trabalhado.

As amostras escolhidas são de *Kernel* ou Usuário/Aplicação. Foi limitado-se a essas duas camadas, pois a capacidade de verificação dos programas de detecção atuais não conseguem transpor além do *Kernel*, e também por causa da segurança, uma vez que um código funcional de *rootkit* de hipervisor ou de firmware/hardware ultrapassariam as limitações da máquina virtual e poderiam causar danos ao equipamento de teste, além do perigo de disseminação para outras máquinas.

4.2.2 Testes

Ao todo foram encontrados 53 (cinquenta e três) repositórios com códigos de *rootkits* para sistemas Unix, com a última modificação datando de 2014 (quatro repositórios), 2015 (um repositório) e algum mês de 2016 (catorze repositórios). Para cada um desses exemplares, foi criada uma máquina virtual usando o programa VMWare com Ubuntu 16.04 instalado, e dado o nome do usuário que a desenvolveu.

As amostras foram compiladas via *Makefile* ou diretamente com o GCC. Ao final da bateria de testes, 19 (dezenove) dos 53 (cinquenta e três) funcionaram no Ubuntu 16.04. Dentro desses 19 (dezenove) repositórios que tinham códigos funcionais, existem 39 (trinta e nove) amostras, sendo que 38 (trinta e oito) são de *rootkits* de *Kernel* e 1 (um) de Usuário/Aplicação.

4.3 Instalação

4.3.1 Na camada de Usuário/Aplicação

- **Na camada de Usuário/Aplicação**
  Para compilar a amostra que funcionou na camada de usuário, bastou compilar com *Makefile* ou diretamente usando o GCC para gerar o arquivo executável do programa. Para usa-lo, rodou-se o executável como qualquer outro programa do Linux, como mostrado no Código 4.1.

Listing 4.1: Comando de Execução

```
1 make
2 ./<nome_do_programa>
```

- **No Kernel**
  Para compilar as amostras que funcionam no *Kernel*, executou-se o *Makefile* para gerar todos os arquivos necessários para a instalação, incluindo o .ko, que é um LKM ou um Modulo Carregado do *Kernel*, então efetuou-se o comando *insmod*, que insere esse novo módulo dentro do *Kernel*. O passo-a-passo está demonstrado no Código 4.2.

Listing 4.2: Comando de Execução

```
1 make
2 sudo insmod <nome_do_modulo.ko>
```

4.4 Amostras

Todas as amostras de *rootkit* foram chamadas pelos *nicknames* que seus autores utilizam no GitHub, pois para muitas delas não foi dado um nome, e assim para evitar confusões e deixar mais organizado foi adotada esta prática.

- **ah450** [Ismail, 2014]
  - Nome: Não informado
  - Tipo: *Kernel*
  - Versão do *Kernel* Testado: >= 3.0
  - Documentação: Não fornecida
- **Arciryas** [Arciryas, 2016]
  - *Sample 1*
    - Nome: Não informado
    - Tipo: *Kernel*
    - Versão do *Kernel* Testado: Não Informado
    - Documentação: Não fornecida
  - *Sample 2*
    - Nome: Não informado
    - Tipo: *Kernel*

    - Versão do *Kernel* Testado: Não Informado
    - Documentação: Não fornecida

- **cloudsec** [wzt, 2015]
  - Nome: Brootkit
  - Tipo: *Kernel*
  - Versão do *Kernel* Testado: Não Informado
  - Documentação: Consultar **Apêndice A.1**
- **Davodu** [Odu, 0016]
  - Nome: Não Possui
  - Tipo: *Kernel*
  - Versão do *Kernel* Testado: Não Informado
  - Documentação: Não informado
- **deb0ch** [de Beauchêne, 2016]
  - Nome: Toorkit
  - Tipo: *Kernel*
  - Versão do *Kernel* Testado: Não Informado
  - Documentação: Não informado
- **klampworks** [Lampis, 2016]
  - *Sample 1*
    - Nome: Muriel e6.4
    - Tipo: *Userland*
    - Versão do *Kernel* Testado: Não Informado
    - Documentação: Não informado
- **leurfete** [Phillips, 2016]
  - Nome: Não informado
  - Tipo: *Kernel*
  - Versão do *Kernel* Testado: Não Informado
  - Documentação: Consultar **Apêndice A.2**
- **m0nad** [Mello, 2016]
  - Nome: Diamorphine
  - Tipo: *Kernel*
  - Versão do *Kernel* Testado: 2.6.x/3.x/4.x
  - Documentação: Consultar **Apêndice A.3**
- **miagilepner** [miagilepner, 2016]

  - Nome: Porny
  - Tipo: *Kernel*
  - Versão do *Kernel* Testado: Não informado
  - Documentação: Não informado

- **mncoppola** [Coppola, 2014]
  - Nome: Suterusu
  - Tipo: *Kernel*
  - Versão do *Kernel* Testado: 2.6/3.x
  - Documentação: Consultar **Apêndice A.4**

- **NoviceLive** [Zhengxiong, 2016]
  - *Experimento 1*
    - Nome: fshid
      - Tipo: *Kernel*
      - Versão do *Kernel* Testado: 4.x
      - Documentação: Consultar **Apêndice A.5**
    - Nome: fsmon
      - Tipo: *Kernel*
      - Versão do *Kernel* Testado: 4.x
      - Documentação: Consultar **Apêndice A.5**
    - Nome: hello
      - Tipo: *Kernel*
      - Versão do *Kernel* Testado: 4.x
      - Documentação: Consultar **Apêndice A.5**
    - Nome: psmon
      - Tipo: *Kernel*
      - Versão do *Kernel* Testado: 4.x
      - Documentação: Consultar **Apêndice A.5**
    - Nome: sys_call_table
      - Tipo: *Kernel*
      - Versão do *Kernel* Testado: 4.x
      - Documentação: Consultar **Apêndice A.5**
  - *Experimento 2*
    - Nome: fshid
      - Tipo: *Kernel*
      - Versão do *Kernel* Testado: 4.x
      - Documentação: Consultar **Apêndice A.5**
    - Nome: kohid
      - Tipo: *Kernel*
      - Versão do *Kernel* Testado: 4.x

        - Documentação: Consultar **Apêndice A.5**
    - Nome: komon
        - Tipo: *Kernel*
        - Versão do *Kernel* Testado: 4.x
        - Documentação: Consultar **Apêndice A.5**
    - Nome: pshid
        - Tipo: *Kernel*
        - Versão do *Kernel* Testado: 4.x
        - Documentação: Consultar **Apêndice A.5**
    - Nome: pthid
        - Tipo: *Kernel*
        - Versão do *Kernel* Testado: 4.x
        - Documentação: Consultar **Apêndice A.5**
    - Nome: root
        - Tipo: *Kernel*
        - Versão do *Kernel* Testado: 4.x
        - Documentação: Consultar **Apêndice A.5**
- *Experimento 3*
    - Nome: codeinj
        - Tipo: *Kernel*
        - Versão do *Kernel* Testado: 4.x
        - Documentação: Consultar **Apêndice A.5**
    - Nome: elf
        - Tipo: *Kernel*
        - Versão do *Kernel* Testado: 4.x
        - Documentação: Consultar **Apêndice A.5**
    - Nome: noinj
        - Tipo: *Kernel*
        - Versão do *Kernel* Testado: 4.x
        - Documentação: Consultar **Apêndice A.5**
    - Nome: real
        - Tipo: *Kernel*
        - Versão do *Kernel* Testado: 4.x
        - Documentação: Consultar **Apêndice A.5**
- *Experimento 4*
    - Nome: get
        - Tipo: *Kernel*
        - Versão do *Kernel* Testado: 4.x
        - Documentação: Consultar **Apêndice A.5**
    - Nome: ifmon
        - Tipo: *Kernel*

            - Versão do *Kernel* Testado: 4.x
            - Documentação: Consultar **Apêndice A.5**
        - Nome: rec
            - Tipo: *Kernel*
            - Versão do *Kernel* Testado: 4.x
            - Documentação: Consultar **Apêndice A.5**
        - Nome: set
            - Tipo: *Kernel*
            - Versão do *Kernel* Testado: 4.x
            - Documentação: Consultar **Apêndice A.5**
    - *Experimento 5*
        - Nome: jmp
            - Tipo: *Kernel*
            - Versão do *Kernel* Testado: 4.x
            - Documentação: Consultar **Apêndice A.5**

- **nsarlin** [nsarlin, 2014]
    - Nome: Não informado
    - Tipo: *Kernel*
    - Versão do *Kernel* Testado: Não informado
    - Documentação: Não informado
- **nu11secur1ty** [nu11secur1ty, 2016]
    - Nome: Não informado
    - Tipo: *Kernel*
    - Versão do *Kernel* Testado: Não informado
    - Documentação: Consultar **Apêndice A.6**
- **QuokkaLight** [Light, 2016]
    - Nome: rkduck
    - Tipo: *Kernel*
    - Versão do *Kernel* Testado: 4.x
    - Documentação: Consultar **Apêndice A.7**
- **soad003** [soad003, 2016]
    - Nome: Não informado
    - Tipo: *Kernel*
    - Versão do *Kernel* Testado: Não informado
    - Documentação: Não informado
- **t0t3m** [t0t3m, 2014]

  - Nome: AFkit
  - Tipo: *Kernel*
  - Versão do *Kernel* Testado: 3.14.1
  - Documentação: Consultar **Apêndice A.8**

- **TengHu** [Hu, 2016]
  - Nome: Não informado
  - Tipo: *Kernel*
  - Versão do *Kernel* Testado: Não informado
  - Documentação: Não informado

- **xuqicx23** [Xu, 2016]
  - Nome: Não informado
  - Tipo: *Kernel*
  - Versão do *Kernel* Testado: Não informado
  - Documentação: Não informado

- **zoeyqi** [zoeyqi, 2016]
  - Nome: Não informado
  - Tipo: *Kernel*
  - Versão do *Kernel* Testado: Não informado
  - Documentação: Consultar **Apêndice A.9**

4.5 Escolha das Amostras

Com o espectro de amostras em um número grande (forma 39 exemplares encontrados), foi necessário fazer uma filtragem com as seguintes diretivas: (1) Entre as amostras observadas foram encontrados alguns projetos muito semelhantes, então optou-se por escolher aquela que tivesse a melhor documentação; (2) Nos repositórios com mais de uma amostra, analisou-se todas e foi escolhida aquela que faria algo diferente do já proposto pelas demais as amostras já selecionadas. A seleção final pode ser observada na Tabela 4.1.

4.6 Conclusão

Encontrar amostras de *rootkits* é um desafio significativo mesmo nesta era onde é possível obter tudo via internet, e isso mesmo para amostras do sistema Windows. A razão para isso são embasadas em inúmeras suposições, entre elas: (1) Desenvolver códigos, realmente bons, para *rootkits* demanda tempo; (2) São códigos de grande valor para serem disponibilizados gratuitamente; (3) Quem os utiliza são, em grande maioria criminosos cibernéticos, e por isso caso alguma empresa de anti-malware conseguisse o seu código, poderia prevenir seu ataque. Pode ser que sejam todas essas suposições correlacionadas, mas não deve haver somente uma resposta para esta questão.

Amostra	Nome	Tipo	Versão do *Kernel* Testado	Documentação
cloudsec	Broodkit	*Kernel*	Não Informado	Apêndice A.1
deb0ch	Tootkit	*Kernel*	Não Informado	Não Informado
klampworks	Muriel e6.4	*Userland*	Não Informado	Não Informado
leurfete	Não Informado	*Kernel*	Não Informado	Apêndice A.2
m0nad	Diamorphine	*Kernel*	Não Informado	Apêndice A.3
miagilepner	Porny	*Kernel*	Não Informado	Não Informado
NoviceLive	Exp5 - jmp	*Kernel*	4.x	Apêndice A.5
QuokkaLight	rkduck	*Kernel*	4.x	Apêndice A.7
t0t3m	AFkit	*Kernel*	3.14.1	Apêndice A.8
zoeyqi	Não Informado	*Kernel*	Não Informado	Apêndice A.9

Tabela 4.1: Tabela dos *rootkits* escolhidos para a experimentação.

Um fato interessante foi observado em relação à instalação das amostras de *rootkits de Kernel*, foi extremamente simples de se fazer, o que demonstra uma falha de segurança em relação à instalação de LKM, pois se um atacante já estiver com a máquina sob seu poder, ele facilmente poderá instalar qualquer módulo malicioso no sistema. A solução ótima para o problema seria não ter módulos, mas como isso não é possível, o que pode ser feito para ajudar a amenizar o problema é exigir que todos os módulos sejam assinados. A assinatura ajudaria a evitar que módulos não assinados fossem instalados, impedindo que códigos maliciosos sejam colocados na máquina.

Capítulo 5

Programas de Detecção

5.1 Introdução

Este capítulo irá apresentar os programas de detção de *rootkits* voltados para sistema Linux que foram escolhidos para esta avaliação. Inicialmente serão apresentadas as motivações que levaram à escolha desses softwares e a metodologia de avaliação empregada. Em seguida, serão apresentados os softwares, algumas informações consideradas pertinentes sobre eles, as respectivas avaliações e, por fim, as conclusões a que se chegou após a avaliação preliminar dos programas.

5.2 Diretivas

5.2.1 Motivação

Existem vários programas circulando pela internet que podem ser usados para detecção de *rootkits* dentro do sistema. Por este fato tornou-se necessário encontrar pontos que pudessem ser satisfeitos e ajudassem a selecionar entre 5 (cinco) e 15 (quinze) programas para serem posteriormente avaliados em sua eficácia de detecção, usabilidade, interface e tempo de análise.

Para a escolha dos programas foram usados os seguintes critérios:

- Ocorrência em artigos acadêmicos sobre o assunto abordado.
- Pesquisa entre usuários do SO escolhido.
- Popularidade nos resultados do *Google* (primeira página).
- Ser gratuito.
- Recorrência em *top* 10 (dez) em sites de ferramentas/informativos.

5.2.2 Avaliação

Os programas escolhidos sofrem duas etapas de avaliação: (1) Uma avaliação inicial, onde são submetidos à análise de usabilidade e interface, (2) Uma avaliação em tempo de experimentação, onde serão observados a eficácia dos métodos de detecção[1] e o tempo decorrido para cada análise.

[1]Ou a capacidade daquele método detectar a amostra de *rootkit* instalada.

Para esta primeira etapa, serão definidos níveis de dificuldade em função de usuários que estariam aptos a utilizar estes programas. Os softwares também serão avaliados em questão de interface do programa, através de uma análise que buscará, principalmente, demonstrar a clareza da passagem de informações ao usuário, o quão agradável/amigável são as telas dos *websites*, como são apresentadas as mensagens de erro, e sobre o material disponibilizado pelo(s) desenvolvedor(es).

5.3 Programas

5.3.1 AIDE

- **Nome:** Advanced Intrusion Detection Environment - AIDE
- **Desenvolvido por:** Rami Lehti, Pablo Virolainen[2], Richard van den Berg[3] e Hannes von Haugwitz[4]
- **Tipo de Detecção:** De Integridade.
- **Interface:** Por linha de comando.
- **Data da Última Atualização:** 25/07/2016
- **Versão:** 0.15.1
- **Avaliação de Material Web:**
  Na *homepage* o site é de layout simples, com várias informações de diversos tipos separadas por títulos e subtítulos em negrito. Não existe uma real separação dos conteúdos, sendo todos eles colocados em sequência com pouco cuidado. Existem duas formas de se instalar o AIDE: (1) apt-get ou (2) baixando o *source code*, mas essas informações se encontram em lugares discrepantes, uma vez que uma delas está na *homepage* e a outra se encontra na página do manual. Embora a parte do manual seja bem explicada e sequenciada, talvez fosse mais interessante, para o usuário, ter um passo-a-passo de como instalar e configurar o programa, pois todos os passos estão separados por tópicos e os comandos estão em meio aos textos e sem uma formatação que chame a atenção, se o usuário for desatento, irá ignorá-los e procurar na internet, formas de utilizar o programa. É um site que dispõe de muita informação, mas não consegue distribuí-la de uma forma mais agradável, a própria interface do site não chama muita atenção e em alguns caso compromete a passagem de informação para o usuário.
- **Avaliação de Material Local:**
  O AIDE dispõe de dois manuais: (1) *man*, que tem documentado seus comandos, parâmetros, diagnostico, principais arquivos, *bugs* conhecidos; e (2) *man* do aide.conf, que auxilia no desenvolvimento do arquivo de configuração, para que o software consiga cumprir com o seu propósito. Estes dois *mans* tornam mais claros e melhor explicados o conteúdo sobre o programa e formas de uso.

[2]Foram os idealizadores do AIDE em 1999.
[3]Fez a manutenção do projeto de 2003 a 2010.
[4]Desde 2010 está cuidando do projeto.

- **Avaliação de Interface de Aplicação:**
  O AIDE possui uma interface por linha de comandos, assim sendo, todos os resultados são escritos na tela do terminal. Quando o programa executa um *–check ou –update*, algumas informações como o *"Changed Entries"*, torna-se mais compreensíveis quando se consulta o *man aide.conf*. O arquivo de configuração é de extrema importância para que se compreenda melhor os resultados, pois é com base nele que o programa gera o banco de dados para fazer as suas análises. Por este fato, é aconselhável manter este arquivo seguro, pois se um programa malicioso alterá-lo a verificação do computador estará comprometida. Por isso, é fortemente indicado que ele seja guardado fora do computador, em outro dispositivo.

- **Avaliação de Tipo de Usuário:**
  Seria recomendado usar este software: Expert, Avançado e Intermediário. Os dois níveis superiores por deterem o conhecimento necessário para gerar bons arquivos de configuração para usar o programa com uma alta capacidade, afim de achar ocorrências de modificações em partes pertinentes do SO. O Intermediário irá encontrar alguns desafios e talvez não consiga configurar o AIDE da melhor forma possível, mas pode ainda alcançar bons resultados. Aos dois níveis mais baixos (Iniciante e Novato) não é aconselhável a utilização deste programa, pois exige um nível de conhecimento que nenhum desses dois tipos de usuário possui e sendo passível de que o resultado acabe não sendo bem compreendido.

5.3.2 CHKRootkit

- **Nome:** CHKRootkit
- **Desenvolvido por:** Pangéia Informática
- **Tipo de Detecção:** De Assinatura
- **Interface:** Por linha de Comando
- **Data da Última Atualização:** 30/10/2016
- **Versão:** 0.51
- **Avaliação de Material Web:**
  O site é bem organizado e possuí várias informações relevantes sobre a área de atuação, empresa, entre outros, mas tudo bem distribuído no formato de páginas. A interface web tem um *layout*, que apesar de ser simples, é harmonioso, não prejudicando e leitura. Porém existem alguns problemas como: (1) codificação em relação aos caracteres com acento, pois o site não foi configurado par UTF-8, então as letras acentuadas são trocadas por um carácter de ?, atrapalhando a leitura dos textos; (2) Mistura de línguas, uma parte do site está em português, mas uma aba específica contém inúmeros textos em inglês, seria interessante possuir uma versão para cada língua; (3) Ausência de informações detalhadas sobre os softwares disponibilizados.

- **Avaliação de Material Local:**
  O manual é bem escasso em questão de descrição de informações, as informações são dadas de uma forma pontual e direta, dando a impressão que o programa é de fácil utilização. A descrição do software, presente no *man*, é bem breve e pobre, poderia ter sido melhor trabalhada, afim de dar informações mais detalhadas sobre a aplicação.

- **Avaliação de Interface de Aplicação:**
  A interface do CHKRootkit é por linha de comando, mas é simples em questão de uso. O programa pode ser executado de duas formas (ambas como super usuário): (1)Passando somente o nome do programa, ele irá fazer uma verificação breve; (2) Passando o nome do programa e uma opção[5] desejada. As saídas podem ser um tanto confusas, quando se usa uma opção em conjunto com o comando de escaneamento, pois mais informações são escritas na tela e não a um alinhamento entre os arquivos/diretórios e a resposta da análise individual que o programa faz. Já na execução sem opção, as respostas da verificação individual está alinhada com a operação feita pelo programa. Não foi possível causar erros para fazer a verificação de mensagens de erro.

- **Avaliação de Tipo de Usuário:**
  Por sua facilidade de uso, o CHKRootkit poderia ser usado por qualquer usuário. Por ser um programa de fácil usabilidade e tendo uma opção padrão que pode ser utilizada, e não depender de configurações externas, somente sendo necessária a leitura do manual, até mesmo aquele que detém o conhecimento mais escasso conseguiria utilizá-lo sem muito esforço. Porém, ele deve ser instalado localmente, e usuários sem experiência podem não saber como instalar o programa corretamente, então é mais aconselhável que seja utilizado por usuários Intermediários, Avançados e Experts.

5.3.3 ClamAV

- **Nome:** Clam Anti Vírus
- **Desenvolvido por:** Cisco Systems, Inc.
- **Tipo de Detecção:** Múltiplas Técnicas[6]
- **Interface:** Por linha de Comando
- **Data da Última Atualização:** 03/05/2016
- **Versão:** 0.99.2
- **Avaliação de Material Web:**
  O site do ClamAV tem toda a informação necessária para passar ao usuário e distribuir seus produtos. Possui um *layout* razoavelmente agradável, mas que poderia ser melhor trabalhado. Na página de *Documentation* não fica bem claro se os textos no início da página são *links* ou apenas textos sobre o conteúdo que poderá ser encontrado naquela página. A grande quantidade de informações e o *layout* escolhido possuem uma certa incompatibilidade, por dar a impressão que os textos foram colocados em lugares quaisquer, tomando cuidado somente para manter os grupos de assuntos concentrados em uma mesma parte do site.

- **Avaliação de Material Local:**
  Em questão de material disponibilizado, o ClamAV possuí dois manuais: (1) *man* clássico, por linha de comando; e (2)Manual do Usuário em PDF, distribuído pelos desenvolvedores. O (1) traz as informações de forma mais pontual, mostrando as opções que podem ser usadas na verificação e alguns exemplos de execução. O (2) tem uma descrição bem mais

[5] O *man* oferece todas as opções que podem ser usadas.
[6] As identificadas foram Heurística (não informado qual) e por Assinatura de Hash.

completa da ferramenta, modos de instalação, *updates*, comandos, de pacotes que são instalados com o ClamAV, dentre outras informações que não foram colocadas ou tiveram a sua descrição feita de forma enxuta no site, além da distribuição do conteúdo no arquivo e a presença de um índice, auxiliem a quem lê encontrar de forma rápida aquilo que lhe é mais pertinente naquele momento.

- **Avaliação de Interface de Aplicação:**
  Por ser uma interface de linha de comando, o ClamAV pode ser algo complicado de se usar no início, mas uma vez lido os materiais de apoio (os dois manuais), é extremamente simples utilizar esta aplicação. Seu método de resposta ao usuário já não é tão agradável, pois ele escreve na tela todo o caminho de cada arquivo que está sendo analisado, e isso é dispensável, bastava ter: (1) somente o nome dos arquivos ou (2)uma informação de qual diretório ele estaria verificando ou (3) a parte mais relevante do caminho. Em contrapartida, ao final da execução, ele traz um conjunto de informações que mostram quantos diretórios foram escaneados, quantidade de dados, quantos arquivos estão infectados, entre outras informações que são convenientes ao usuário saber. Não foram avaliadas mensagens de erro, pois não foi conseguido causar nenhum erro na aplicação.

- **Avaliação de Tipo de Usuário:**
  Com a disponibilização dos dois manuais, principalmente o arquivo em PDF, o ClamAV é um programa que pode ser utilizado por qualquer usuário entre os níveis de Novato a Expert. O programa não exige que do usuário nenhum conhecimento técnico, bastando que este saiba os comandos básicos da aplicação para conseguir utilizá-lo.

5.3.4 Lynis

- **Nome:** Lynis
- **Desenvolvido por:** Michael Boelen[7] e CISOfy development team[8]
- **Tipo de Detecção:** Não Informado
- **Interface:** Por Linha de Comando
- **Data da Última Atualização:** 26/09/2016
- **Versão:** 2.3.4
- **Avaliação de Material Web:**
  O *website* dos desenvolvedores do Lynis é bem dividido e organizado. As informações são separadas de acordo com as competências de cada tópico, no mapa do site. Em questão de *layout*, ele é bem simples, mas garante que o usuário consiga ler as informações e fazer distinção do que é texto e do que é link. Outra característica, que se deve tomar nota, são os tutoriais relacionados a instalação, configuração e como utilizar o programa, o que é muito interessante para usuários que nunca utilizaram o programa. Também é interessante o fato que a cada etapa concluída existe um botão para a próxima, com isso o usuário não precisa ficar procurando o passo seguinte e não perde todo o raciocínio que já vinha fazendo.

[7] 2007 a 2013.
[8] 2013 a -.

- **Avaliação de Material Local:**
  Esta versão do Lynis não é instalável, rodando somente localmente dentro da máquina, então não é possível acessar o *man* usando o comando normal, deve-se executar *"lynis –man"*. Nessa versão do manual, encontram-se as informações sobre o Lynis de uma foram mais breve, porém contém desde a descrição à mensagens de erro. É um material bem completo, e de fácil leitura e interpretação. Existe também o seguinte comando *"lynis –help"*, que mostra somente os comandos disponíveis para a execução do Lynis e as opções possíveis, de forma mais resumida, para usuários ociosos ou de pouca paciência é uma opção bm interessante e facilita o uso.

- **Avaliação de Interface de Aplicação:**
  O Lynis utiliza a interface por linha de comando, por isso escreve toda a saída do programa na tela do terminal para o usuário. Sua saída é feita de uma forma bem organizada (ou a maior parte dela), pois divide o computador em módulos e o que foi escaneado dela, mas mesmo assim torna um pouco complicado do usuário saber qual é o problema pelo qual o seu computador esta passando. O programa também possuí duas formas de verificação: (1) normal, executa uma varredura mais básica no sistema e avisa que a opção executada é sem privilégios, e (2) com *sudo*, que é feita uma verificação mais demorada e mais meticulosa. Não foi possível causa erros no programa, desta forma não foi possível avaliar as mensagens de erro.

- **Avaliação de Tipo de Usuário:**
  Apesar de ser um programa fácil de se utilizar lendo o site, o manual e o *help*, o Lynis traz outras dificuldades que impossibilitam que um usuário novato consiga utilizá-lo, isso se deve por alguns fatos: (1) porque a versão do *apt-get* está desatualizada e deve-se se usar a versão do repositório do *Git* ou a disponibilizada no próprio site; (2) Deve-se executar uma sequência de passos para que se possa utilizar o Lynis como um programa instalado dentro da máquina, e para isso deve se executar alguns comandos com privilégios de super usuário; (3) As mensagens de alerta e sugestões dados nos resultados no Lynis devem ser interpretados de forma correta, pois não se deve somente ver e instalar tudo, deve-se ler sobre o que se trara e verificar se aquilo é relevante ou não. Um novato/iniciante provavelmente encontraria muitas dificuldades para a instalação e não teria conhecimento suficiente para interpretar os resultados da forma correta, além de poder causar problemas a si mesmo quanto ao uso dos privilégios de super usuário. Por essas razões, ele é passivo de utilização de Intermediário, Avançado e Expert.

5.3.5 Ossec

- **Nome:** OSSEC
- **Desenvolvido por:** OSSEC Team
- **Tipo de Detecção:** Múltiplas Técnicas
- **Interface:** Por linha de Comando
- **Data da Última Atualização:** 30/06/2016
- **Versão:** 2.9.0 RC3

- **Avaliação de Material Web:**
  Toda a documentação do software fica no site dos desenvolvedores, porém ela está desatualizada desde final de 2015, pois a versão dos manuais e instalação é a 2.8.3 e em alguns casos faz-se a citação da versão 2.9.0, mas não há qualquer indicação que é a versão mais recente ou se é no pacote de *releases*. Em questão de conteúdo, os assuntos estão bem divididos, o manual disponibilizado tem um *link* para todas as seções caso o usuário queira ler um material especifico, mas nas páginas de cada seção/subseção não há um link ou um botão para a seção/subseção seguinte, caso queira-se continuar a leitura, obrigando o usuário a voltar para à página anterior ou do manual, para clicar no link seguinte, e isso faz com o usuário faça um esforço desnecessário e causa quebra de raciocínio. Em termos de conteúdo, o site possui muita informação sobre o uso, mas na base de muita procura e leitura dos links disponibilizados, nenhum tutorial mais simples é apresentado como opção.

- **Avaliação de Material Local:**
  Não possui manual local da aplicação disponível para acesso ou qualquer outra documentação.

- **Avaliação de Interface de Aplicação:**
  O Ossec, após instalado e iniciado, fica monitorando a máquina em tempo real, todos os alertas são escritos em arquivos de *log*, que podem ser lidos pelo usuário no diretório mostrado em /var/ossec/logs/alerts/. Mesmo que, durante a instalação, o usuário já defina algumas regras de execução para o programa (através de perguntas que são feitas durante o processo de instalação), o Ossec permite que novas regras e novos alertas sejam adicionados, podendo assim aumentar o monitoramento. As informações do *log* são escritas de uma maneira ruim, deve-se lê-las e interpretá-las com cuidado para que não haja falsos positivos, quanto a erros e presença de softwares maliciosos. Um erro que foi observado é referente ao acesso aos *logs*, pois a pasta do Ossec não permite acesso de outro usuário que não seja o *Root*, e isso não foi informado na mensagem de erro que é mostrada no Código 5.1.

Listing 5.1: Comando de Execução

```
1 sudo: cd: comando não encontrado.
```

Esta mensagem não passa nenhuma informação do que está errado com o comando, demonstrando que o sistema não estava preparado para informar ao usuário que aquela pasta só deve ser acessada pelo usuário *Root*.

- **Avaliação de Tipo de Usuário:**
  Embora seja um software que aparenta ser de fácil utilização e instalação, para um monitoramento mais amplo é necessário que sejam adicionadas mais regras e alertas, principalmente se o foco for procurar *rootkits* dentro de um sistema. Para se fazer essas adições, deve-se saber quais os locais que devem ser monitorados, quais alertas devem ser ativados e isso demanda conhecimento prévio, dado a estes fatos ele é mais indicado para usuário Intermediários (que irão ter alguma dificuldade, um pouco na tentativa-e-erro, mas conseguiram adicionar o conteúdo necessário), Advanced e Expert.

5.3.6 RKHunter

- **Nome:** Rootkit Hunter

- **Desenvolvido por:** Michael Boelen[9], John Horne[10], unSpawn[11], Aus9[12], Gary Bak[13], Andrej Ricnik[14], konsolebox footnoteNome verdadeiro não divulgado, responsável por pegar sugestões e testes., Sibtay Abbas[15], Constantin Stefan[16], Iain Roberts [17], Doncho N. Gunchev[18], Steph [19]

- **Tipo de Detecção:** Não informado

- **Interface:** Por linha de Comando

- **Data da Última Atualização:** 24/02/2014

- **Versão:** 1.4.2

- **Avaliação de Material Web:**
  Não há muitas informações sobre o RKHunter na página *web* disponibilizada pelos desenvolvedores, somente as consideradas mais pertinentes, como *links* para *downloads*, listas de emails, e algumas poucas informações sobre a equipe de desenvolvimento e FAQ. O *layout* é bem simples e não atrapalha a leitura do usuário. O conteúdo de como utilizar está disponibilizado em uma Wiki, no repositório do *Source Forge*, onde todas as informações sobre usabilidade e comandos estão bem descritas, mas seus *links* poderiam ser disponibilizados ou este conteúdo poderia sido colocados no site diretamente, para evitar o espalhamento de documentação em vários locais, isso confunde o usuário e a página seria melhor utilizada.

- **Avaliação de Material Local:**
  O único material de documentação que realmente traz alguma informação é o *man*, trazendo todas as informações necessárias para que o usuário consiga compreender o que é o RKHunter e como utilizá-lo, seus comandos, opções e o que cada um deles faz. Porém existe uma seção chamada [Tests], que ainda não foi escrita, tendo vários tipos de teste que podem/são/irão ser feitos pela ferramenta, como não a descrição, ela poderia ser ocultada o manual.

- **Avaliação de Interface de Aplicação:**
  O RkHunter tem como interface ser por linha de comando, então suas saídas serão escritas ne tela, outro fato interessante é que ele só executa com privilégio de super usuário, para poder ter acesso a certas pastas onde não se permite execução. Cada linha da saída do comando de varredura do sistema, mesmo aquelas que mostram parte de um caminho que está sendo analisado ou se está analisando a procura de um *malware* específico, são alinhadas com o resultado encontrado, e ao final da execução são mostradas outras informações como quantidade de arquivos verificados, se foi encontrado alguma ocorrência possível de ser um *rootkit*, ou seja, o resultado é de fácil interpretação para o usuário

[9]Devolvedor inicial.
[10]Desenvolvedor atual.
[11]Nome verdadeiro não foi divulgado, desenvolvedor atual.
[12]Nome verdadeiro não foi divulgado, responsável pela documentação.
[13]Melhorar suporte para AIX e Teste.
[14]Manutenção e Teste.
[15]Teste.
[16]Ideias para melhoria.
[17]AIX e suporte para OpenBSD.
[18]Função não divulgada.
[19]Nome completo não divulgado, Testes.

comum o que evita entendimento errado por parte dele. O único erro causado foi a falta de privilegio de super usuário, e a mensagem de erro era condizente com este fato.

- **Avaliação de Tipo de Usuário:**
  Embora seja um programa de fácil instalação e utilização, deve-se baixar o .tar.gz e instalá-lo manualmente, usuário Novatos podem ter dificuldades em saber quando devem instalar o pacote com privilégios de super usuário e quando não devem, e também por não estarem acostumados com os comandos do linux, por isso é mais aconselhável que usuários Iniciantes a Expert utilizem este programa.

5.3.7 Tripwire

- **Nome:** Tripwire Open Source
- **Desenvolvido por:** Tripwire, Inc. [20], mantido por Brian Cox, Thom O'Connor, Paul Herman, David LaPalomento
- **Tipo de Detecção:** De Integridade
- **Interface:** Por linha de comando
- **Data da Última Atualização:** 12/04/2016
- **Versão:** 2.4.2.2
- **Avaliação de Material Web:**
  Tanto no site da Tripwire.org quanto no da Tripwire.com não há qualquer informação sobre o Tripwire OP, a única informação encontrada foi uma pequena nota no site do Tripwire.org sobre o repositório no GitHub, demonstrando um certo descaso com a ferramenta.
- **Avaliação de Material Local:**
  O manual, *man*, é a única documentação disponível que contém as informações de usabilidade. É um material bem completo para o usuário, descrevendo inicialmente cada tipo de verificação, *update* que pode ser feito, como é feita a inicialização do banco de dados. Porém falta informação quanto a produção de um arquivo de configuração especifico pelo usuário, pois existe a possibilidade de fazer uma verificação mais detalhada do sistema,como a verificação de pendrives e outras pastas.
- **Avaliação de Interface de Aplicação:**
  Por ser uma interface que executa através de linha de comando, então qualquer comando que seja dado irá gerar uma escrita de saída. Nenhuma saída gerada pelos comandos pareceu muito amigável ao usuário, principalmente a que diz respeito ao comando que criava o banco de dados, pois foram mostradas várias mensagens de erro no decorrer da execução, seria melhor apresentar um resultado somente ao final dela o ou um *log* sobre os problemas encontrados. O comando de verificação, apresenta esses resultados também durante a execução, mas também apresenta um resultado detalhado da análise feita. O resultado exibe as informações por setores em que executou, existe também a divisão de conteúdo entre o relatório de execução e o relatório de erros encontrados, mas dificilmente sua conclusão de execução consegue ser compreendida por um usuário qualquer (mesmo aqueles com bastante experiencia, mas que não usam este programa comumente). Em

[20] nos anos 2000.

relação aos erros, conseguiu-se causar ao menos três erros: (1) executar sem o privilegio de super usuário, (2) fazer *update* do programa e (3) executar o comando *–test*; para os dois erros iniciais foi mostrada a mesma mensagem de erro demostrada no Código 5.2.

Listing 5.2: Mensagens de Erro Erradas

```
[comando:] tripwire --check
Error: File could not be opened.
Filename: /usr/local/etc/tw.cfg
Permissao negada
Configuration file could not be read.
Exiting...
```

Listing 5.3: Mensagens de Erro Erradas

```
[comando:] sudo tripwire --update
Error: File could not be opened.
Filename:
/usr/local/lib/tripwire/report/tcc-Latitude-E5510
    -20161022-133858.twr
Arquivo ou diretorio nao encontrado
Exiting...
```

A razão pela quais os erros aconteceram ficaram fora do contexto da mensagem, para um usuário iniciante não seria claro o que ele fez de errado. No caso do erro *update*, Código 5.3, ainda é mais difícil de saber o que ocasionou o erro porque foi dado privilegio de super usuário para a execução do comando. Em contrapartida, o terceiro erro cometido foi o único que gerou uma mensagem de erro esperada demonstrada no Código 5.4.

Listing 5.4: Mensagem de Erro Correta

```
[comando:] sudo tripwire --test
Error: Command line parameter missing.
The switch --test requires --email.
Exiting...
Use --help to get help.
```

- **Avaliação de Tipo de Usuário:**
  É um programa que poderia ser usado por usuários novatos e iniciantes, mas a falta de documentação, a dependência no manual, ter que ser instalado localmente e precisar de um arquivo de configuração especial para verificar o programa de uma maneira mais rigorosa, faz o que os usuários mais habilitados para o uso sejam Intermediários, Avançados e Experts.

5.3.8 Tabela Comparativa

Todos os programas foram avaliados em 4 (quatro) quesitos individualmente, porém é necessário que eles sejam comparados entre si, para que se possam ver as diferenças e similaridades entre os softwares avaliados, e isso pode ser visto na Tabela 5.1. A tabela é formada por 8 (oito) colunas, que expressa os dados da seguinte forma:

- **Detecção:** Qual o tipo de detecção que o programa utiliza.
  - *A:* Assinatura

  - *I:* Integridade.
  - *MT:* Múltiplas Técnicas
  - *NI:* Não Informado

- **Interface:** Qual interface ele utiliza:
  - *LD:* Linha de Comando
  - *IG:* Interface Gráfica

- **Ano:** Ano da última atualização disponível para *download*.

- **Material Web:**
  - *****:* Ótimo, o material é completo, a interface é agradável.
  - ****:* Bom, falta pouca documentação, a interface não é tão bem elaborada ou tem algo que poderia ser melhorado.
  - **\:* Regular, falta alguma documentação, mas é possível achar informações relevantes, a interface não é amigável ou não é bem estruturada.
  - **:* Péssimo, falta toda documentação, o site é confuso, a interface não é agradável e compromete muito a leitura.
  - *-:* Não avaliado, material faltante ou ausente.

- **Material Local:**
  - *****:* Ótimo, o material é completo, a interface é agradável.
  - ****:* Bom, falta pouca documentação, a interface não é tão bem elaborada ou tem algo que poderia ser melhorado.
  - **\:* Regular, falta alguma documentação, mas é possível achar informações relevantes, a interface não é amigável ou não é bem estruturada.
  - **:* Péssimo, falta toda documentação, o site é confuso, a interface não é agradável e compromete muito a leitura.
  - *-:* Não avaliado, material faltante ou ausente.

- **Interface de Aplicação:**
  - *****:* Ótimo, bem fácil de usar, os comandos são fáceis encontrar e utilizar, as mensagens de erro são condizentes com os erros causados.
  - ****:* Bom, existe pouca dificuldade de utilização, as mensagens de erro são condizentes com os erros causados, mas pode haver uma ou
  - **\:* Regular, existe certa dificuldade na utilização, mesmo com a leitura dos manuais, as mensagens de erro não são tão boas quanto o esperado.
  - **:* Péssimo, utilização complexa, comandos não são explicados, as mensagens de erro não são condizentes com nenhum erro encontrado.
  - *-:* Não avaliado, não foi possível chegar a uma conclusão quanto a analise da interface do programa .

- **Tipo de Usuário: [of Health, 2016]**

- *EU: Expert User*, é um especialista da área de Computação, capaz de fornecer orientação, solucionar problemas e responder a perguntas relacionadas a essa área de especialização e ao campo onde a habilidade é usada. Cria novos aplicativos e/ou lidera o desenvolvimento de materiais de referência e de recursos para essa competência; demonstra excelência consistente na aplicação dessa competência em vários projetos e/ou organizações;
- *AU: Advanced User*, pode executar as ações associadas a a tarefas computacionais de geração de código, análise de *logs*, entre outros, sem assistência. É reconhecido certamente como "uma pessoa a perguntar"quando as perguntas difíceis levantam-se a respeito desta habilidade; auxilia no desenvolvimento de materiais de referência e de recursos nessa competência.
- *IU: Intermediate User*, é capaz de concluir com êxito tarefas de computação, como redigir códigos e aplicações completas, fazer análise parcial de *logs*, entre outras, conforme solicitado. Ajuda de um especialista pode ser necessário de vez em quando, mas geralmente pode executar as tarefas de forma independente, porém ainda precisa aprimorar conhecimentos ou habilidades;
- *BU: Beginner User*, Usuário iniciante que acabou de começar a utilizar sistemas Linux, espera-se que precise de ajuda para executar qualquer tipo de tarefa e necessite fazer consultas sobre os serviços disponíveis e tutoriais de utilização.
- *N/A:* Não avaliado, não foi possível chegar a uma conclusão de qual usuário é o mais indicado para o uso do programa.

Programa	**Detecção**	**Interface**	**Ano**	**Material Web**	**Material Local**	**Interface da Aplicação**	**Tipo de Usuário**
AIDE	I	LD	2016	**	***	**	EU, AU, IU
ClamAV	MT	LD	2016	**	***	**	Todos
CHKRootkit	A	LD	2016	**	**	**	EU, AU, IU
Lynis	NI	LD	2016	***	***	***	EU, AU, IU
Ossec	MT	LD	2016	**	-	**	EU, AU, IU
RKHunter	NI	LD	2014	*	**	***	EU, AU, IU, BU
Tripwire	I	LD	2016	-	**	***	EU, AU, IU

Tabela 5.1: Tabela de comparação dos programa avaliados.

5.4 Conclusão

Foram escolhidos selecionados 7 (sete) programas para sistemas Linux para serem avaliados de diversas maneiras, mas essa avaliação foi dividida em duas etapas: (1) Avaliação Inicial, para observar a usabilidade, documentação local e web, quais usuários conseguiriam utilizar esses programas; (2) Avaliação de Experimentação, onde será verificado o tempo de execução, efetividade dos métodos de detecção.

Nessa avaliação inicial, percebeu-se à falta de cuidado de muitos desenvolvedores em relação a documentação web, não a uma preocupação com *layout* ou a forma de apresentação do texto, muitas vezes todo o material descrito para se usar a ferramenta se encontra ali, mas está tão disperso e/ou confuso/misturado com outros assunto que torna-se complicado de encontrar. A interface web deveria ser a porta de entrada para chamar a atenção do usuário para aquela ferramenta, mas não está sendo dada essa importância.

Em relação ao material dos manuais e interface de aplicação, a grande maioria deles atende à premissa comum (o formato do *man* e a ideia de digitar um comando e obter uma resposta) de usabilidade dos sistemas Unix, com a qual os usuários (que utilizam SOs deste tipo) já estão acostumados. Nesses quesitos o descuido foi nas mensagens de erro, pois muitas delas não expressavam o que o usuário fez de errado, e se ele não irá tomar consciência do que errou, não tem como descobrir o modo correto.

O número de ferramentas avaliadas é considerado pequeno, porém, foi avaliado que a grande maioria das ferramentas para sistemas Unix, que eram utilizadas na detecção de *rootkits*, ou não são mais encontradas ou seus repositórios, pararam de ser atualizados, e a grande maioria delas antes de 2010. Pode ser que o fato dos *rootkits* não aparecerem mais na mídia comercial seja correlacionado com a falta de manutenção desses repositórios, o que levou a descontinuação dos projetos. Outra suposição seria que a complexidade e as técnicas de detecção estavam demandando muito tempo e esforço dos desenvolvedores, que não conseguiram dar conta e abandonaram os projetos.

Capítulo 6

Análise de Comportamento

6.1 Introdução

Este capítulo irá apresentar todos os detalhes sobre o experimento que foi, como foi e onde foi realizado. Inicialmente, são informadas as características do hardware e software da máquina e das VMs. Em seguida, haverá a explicação de como os experimentos foram feitos e quais os resultados foram obtidos deles. Por fim, será feita uma consideração envolvendo cada um dos 7 (sete) programas utilizados nas análises e uma conclusão final do experimento.

6.1.1 Máquinas usadas no experimento

6.1.1.1 Máquina Física

Características de Hardware e Software

- **Processador:** Intel Core i5, com 4 (quatro) núcleos.
- **Caches:**
  - *L1*, com tamanho 4x 32KB, 8 way
  - *L2*, com tamanho 4x 256KB, 8 way
  - *L3*, com tamanho 6144KB, 12 way;
- **Placa-Mãe:** Lenovo, com *chipset Intel Corporation*.
- **Memória RAM:** 2GB
- **Sistema Operacional:**
  - *Kernel:* Linux 4.4.0-36-generic,
  - *Distribuição:* Ubuntu 16.04.1 LTS.

6.1.1.2 VMs

Características de Hardware e Software

- **Processador:** Intel Core i5, com 1(um) núcleo.
- **Caches:**

  - *L1*, com tamanho 4x 32KB, 8 way
  - *L2*, com tamanho 4x 256KB, 8 way
  - *L3*, com tamanho 6144KB, 12 way;

- **Placa-Mãe:** *Intel Corporation,chipset Intel Corporation*.
- **Memória RAM:** 1GB
- **Sistema Operacional:**
  - *Kernel:* Linux 4.4.0-36-generic,
  - *Distribuição:* Ubuntu 16.04.1 LTS.

6.1.2 Diretivas do Experimento

6.1.2.1 Sistema Operacional

O sistema Operacional escolhido foi o Ubuntu 16.04 por ser um dos sistemas mais comumente utilizados por usuários de Sistemas Linux iniciantes ou aqueles que estão migrando de sistema, por ele ser um dos que tem interface mais amigável e ser de fácil manuseio.

6.1.2.2 Ambiente de Experimentação

Foram feitas 10 (dez) máquinas virtuais no programa VMWare[1] seguindo a configuração **Seção 6.1.1.2**. Em cada uma das máquinas foi instalada uma única amostra para que não houvesse problemas de uma amostra de *rootkit* inutilizar a outra e para manter a integridade do sistema hospedeiro, limitando as modificações desse único *rootkit*.

Foi optado fazer uso de máquinas virtuais por elas permitirem, de uma maneira mais facilitada no *host*, fazer o isolamento das amostras, não permitindo que outros computadores fossem contaminados ou que elas se espalhassem pela rede.

6.1.2.3 Procedimento de Análise

A fase de análise foi dividida em duas partes, para se obter um resultado mais amplo da capacidade de detecção dos programas que foram escolhidos. Essas duas fases são:

- **Fase 1 - Análise Externa/Offline**
  Nesta primeira fase, as máquinas serão "montadas", como se fossem pendrives ou HD's externos, e analisadas a partir do *host*. Esta análise é a mais confiável, pois como as máquinas infectadas estão desligadas e estão sendo analisadas de fora, não tem como o *rootkit* ou qualquer *malware* manipular/modificar os logs e laudos das análises.

- **Fase 2 - Análise Interna/Online**
  Nesta segunda fase, as máquinas virtuais serão ligadas e analisadas como se fossem sistemas reais, ou seja, os programas serão executados dentro das VMs comprometidas. Este tipo de análise não é confiável, pois o *rootkit* pode manipular arquivos de configurações ou arquivos sensíveis dos programas de detecção e modificar seus resultados para ocultar a sua presença, tornando o resultado da verificação incorreto.

[1] Foi escolhido este programa por ele facilitar a transferência de arquivos da máquina *host* para a máquina virtual, principalmente através da montagem automática do pendrive.

6.2 Experimentos

Os experimentos foram feitos em duas etapas, para ser feito um comparativo dos resultados que os programas de detecção obtiveram. Essas duas etapas são descritas como (1) Fase 1 - Análise Externa ou Offline e (2) Fase 2 - Análise Interna ou Online, que foram definidas na **Seção 6.1.2.3**.

6.2.1 Fase 1 - Análise Offline

Todas as VMs foram "montadas" no computador *host* somente com o SO instalado, para que os programas de integridade e/ou os que necessitam de registros conseguissem gerar os bancos de dados com os arquivos das VMs. Este passo é de extrema importância, pois os bancos de dados irão guardar *hashes* de cada um dos arquivos contidos nas máquinas virtuais, então se algum dos arquivos sofrer alterações e for constado na análise, será apontada uma diferença em relação ao dado salvo. Os arquivos de configuração foram editados para que conseguissem acessar o diretório onde as máquinas estavam montadas.

Depois que todos os bancos de dados foram inicializados, as máquinas virtuais foram "desmontadas" do sistema *host* e executadas sequencialmente para a instalação das amostras em cada uma delas. Foi instalada apenas uma única amostra de *rootkit* em cada VM, para que o sistema não fosse comprometido de uma forma abrupta e os exemplares não se auto-contaminassem e/ou se auto-destruíssem e/ou comprometessem o experimento.

Com todas as amostras instaladas em suas respectivas máquinas, cada uma delas foi desligada ou o seu estado foi salvo, com o *rootkit* ainda em atividade, pois alguns deles não eram persistentes na máquina e paravam de executar assim que esta era desligada. Então, as imagens das VMs foram novamente "montadas" no sistema de arquivos do *host* e verificadas com cada um dos programas de execução.

6.2.2 Fase 2 - Análise Online

Foram feitas novas máquinas virtuais e feita uma nova instalação do Ubuntu 16.04. Foi adotado este procedimento para que não houvesse perigo de contaminação do *host* por parte de algumas amostras instaladas durante a **Fase 1**. Nestas imagens limpas, foram colocados os arquivos *.tar.gz* dos programas de detecção e os *.zip* dos *rootkits* escolhidos.

Em cada VM, foram instalados os programas de detecção e um único *rootkit*, pelos motivos já citados anteriormente na **Seção 6.2.1**. Não há preocupação na instalação de mais de um software de detecção na mesma máquina, pois eles não fazem a remoção das amostras e eles não causam conflito entre si, ou seja, permitem que mais de um software deste tipo seja instalado na mesma máquina sem causar danos ao sistema.

Com as amostras e os programas instalados, cada uma das ferramentas foi executada para fazer a verificação do sistema da máquina virtual.

6.3 Resultados

6.3.1 Fase 1 - Análise Offline

Por ordem alfabética de nomes, foram executados os programas de detecção com as VMs "montadas" na máquina *host*. Foi executado esta maneira por facilitar a visualização dos resultados, que serão listados a seguir.

- **AIDE:**
  O tempo de duração da verificação foi cerca de 5 (cinco) horas e 30 (trinta) minutos, um tempo longo para o usuário esperar, mas que reflete que a ferramenta esta verificando todas as pastas e arquivos. Conforme a análise é feita, é impresso na tela o que foi encontrado de modificações em relação ao banco de dados do próprio programa. O AIDE detectou diferenças em 9 (nove) das 10 (dez) pastas, mas não é informado a natureza dessas modificações para o usuário, ou seja, não se sabe se essas alterações foram feitas devido a alguma atualização ou se é algo malicioso.

- **CHKRootkit:**
  O tempo de duração da verificação foi de 29 (vinte e nove) minutos no total, pois cada VM foi verificada por vez. O tempo, de análise por VM, foi de 1 (um) minuto e 37 (trinta e sete) segundos a 4 (quatro) minutos e 7 (sete) segundos, são tempos curtíssimos, e reflete que a simplicidade da análise feita. A investigação do CHKRootkit é divida em duas formas, uma procura inicial por alterações em comandos e uma procura final por *rootkits* que tenham assinaturas presentes em seu banco. Nenhuma das 10(dez) amostras foi localizada, o motivo para estes resultado é que este programa além da sensibilidade provida pelo sistema de verificação ser de assinatura é que ele atua na em *userland* ou camada de usuário, e como a maioria das amostras são de *Rootkits de Kernel*, ele não foi capaz de detectar.

- **ClamAV:**
  O ClamAV é uma ferramenta de antivírus, e detecta várias ameaças maliciosas, uma vez que o usuário comum não saiba o porque seu sistema estar apresentando características estranhas, como lentidão, travadas, programas finalizando de forma não esperada, irá utilizar uma ferramenta como essa, por ser uma das mais conhecidas, e foi este fato que o incluiu nas ferramentas de análise, para testar se qual o seu comportamento frente a uma ameaça de difícil de detecção e um tipo de ameaça crescente. O tempo de análise foi de 1(uma) hora e 53 (cinquenta e três minutos), um tempo longo, e na tela é mostrado o caminho de cada arquivo que esta sendo verificado. Nenhuma das 10(dez) amostras foi localizada, acreditava-se que ao menos uma poderia ser localizada e classificada como sendo uma outra ameaça, como um *Trojan*, como comumente acontece, mas isso não ocorreu.

- **Lynis:**
  A análise do Lynis é feita em forma de módulos, então cada parte do sistema é analisado em um módulo, alguns deles são *Networking, Ports and packages, Files systems*, entre outros. É um modo interessante, mas não auxilia realmente para que o usuário compreenda o que ele pode fazer em relação aos resultados que não são positivos. O tempo de análise foi considerado muito curto, 2 (dois) minutos e 30 (trinta) segundos e nenhuma das 10(dez) amostras foi localizada. Com estes resultados negativos e o curto tempo, foi identificado, através de pesquisas feitas, que o Lynis não consegue verificar sistemas de outros computadores, somente do sistema em que está instalado, o que é uma falha grave para softwares detecção, pois não existe confiança em uma análise feita dentro de um sistema comprometido.

- **OSSEC:**
  As análises do OSSEC ocorrem em tempo real, pois ele monitora o sistema o tempo todo e gera um alerta para cada modificação, desde um simples login a alguma modificação. Foram detectadas diferenças em 9 (nove) das 10 (dez) pastas das VM, mas nenhum alerta de *Rootkit* foi gerado, o que indica que só foram gerados alertas em relação às modificações

feitas nos arquivos. Com algumas pesquisas, identificou-se que o OSSEC, assim como o Lynis, não consegue escanear sistema de arquivos de outro computador.

- **RKHunter:**
  O tempo de verificação foi de 1 (um) minuto 10 (dez) segundos, um tempo considerado bem curto para uma análise de sistema completa. A análise do RKHunter também é dividida em módulos, mas é uma análise voltada completamente para a busca de alterações comuns de *rootkits* no sistema, e em conjunto também é feita uma busca por certos tipos de *rootkits* que estão presentes no banco de dados do programa. Nenhuma das 10(dez) amostras foi localizada, e este resultado negativo foi atribuído a duas razões: (1) o RKHunter não está sendo atualizado há muito tempo, a justificativa de seus desenvolvedores é justamente a falta de tempo para se dedicar ao projeto, e como ocorre de muitos softwares não terem incentivo monetário para auxiliar em seu desenvolvimento, o projeto acaba tornando-se secundário; e (2) Ele também não faz verificação externa, ou seja, não verifica um sistema de outro computador.

- **Tripwire:**
  O tempo de verificação foi de 2 (duas) horas e 10 (dez) minutos. Embora apresente como resultado, várias mensagens que não seriam interessantes para o usuário observar, o Tripwire mostra na tela um resultado modular, mas que não demonstra ser algo que o usuário compreenderia ou que ele tem o conhecimento do que seja. Nenhuma das 10(dez) amostras foi localizada, e este resultado negativo foi atribuído a ele também não faz verificação de sistemas de outros computadores, que não seja o que ele está instalado.

Nenhum dos programas de detecção conseguiu identificar o *rootkit de userland*, pois sua execução é muito rápida e as modificações só persistem enquanto o *rootkit* está em execução, foi tentado salvar o estado da VM, mas não obteve-se sucesso a ser notado. Uma análise comparativa dos resultados pode ser observada na Tabela 6.1. Para a avaliação dos resultados foram usadas as seguintes diretivas:

6.3.1.1 Avaliação dos Resultados

- **Qtde de Diferenças** Quantidade de diferenças localizadas após a instalação da amostra.
- **Resposta da Análise:**
  - *****:* Ótimo, as mensagens escritas na tela e o *log* pode ser compreendidos perfeitamente pelo usuário, ou seja, não é necessário conhecer siglas ou outros símbolos, nem é difícil de interpretar os resultados.
  - ****:* Bom, as mensagens e o *log* necessitam de um pouco de conhecimento para serem compreendidas, mas basta olhar o *man* ou alguma documentação do programa e pode-se entender como o programa classificou os resultados e de que forma.
  - **\:* Regular, exige que o usuário análise com certo cuidado todas as suas saídas, para que assim, ele consiga compreender o que o programa fez e o que foi analisado.
  - **:* Péssimo, as mensagens escritas na tela e o *log* não passam nenhuma informação que o usuário, de forma, que seja passível de se interpretar.
  - *-:* Não avaliado, não foi possível fazer avaliação.
- **Tempo:**

- ****:* Ótimo, 1 hora ou mais, análise extremamente demorada.
- ***:* Bom, até quarentena minutos, análise parcialmente demorada.
- **:* Regular, de vinte a trinta minutos, análise com duração intermediária.
- *:* Péssimo, menos de vinte minutos, análise extremamente curta.
- *-:* Não avaliado, não foi possível avaliar o tempo.

Programa	Qtde de Diferenças	Resposta da Análise	Tempo
AIDE	9	-	****
CHKRootkit	0	*	**
ClamAV	0	**	****
Lynis	0	-	*
Ossec	9	**	-
RKHunter	0	***	*
Tripwire	0	-	****

Tabela 6.1: Tabela de comparação dos programas avaliados.

6.3.2 Fase 2 - Análise Online

Novamente a execução foi por ordem alfabética, pelo mesmo motivo da **Seção 6.3.1**. Porém, todos os programas de detecção foram executados dentro das máquinas virtuais onde os *rootkits* foram instalados. Este experimento foi um pouco diferente do anterior também porque os detectores foram instalados antes das amostras de *rootkits* para simular um ambiente real, uma vez que é bastante comum usuários possuírem softwares de segurança e mesmo assim serem infectados. Outro diferencial é que foram usadas somente 8 (oito) das 10 (dez) amostras, pois uma delas era a *userland*, **klampworks**, que não tinha uma instalação persistente, sendo uma única execução, e a amostra **deb0ch**, pois ela apaga a *libselinux.so.1* e, com isso, causa-se um erro ao tentar abrir as bibliotecas compartilhadas e torna-se impossível de se navegar nos diretório, impedindo que fossem acessados os programas de detecção, ou desligar o sistema.

- **AIDE:**
  Neste segundo teste, os tempos de verificação alternaram entre 30 (trinta) a 50 (cinquenta) minutos. Como esperado, o resultado da verificação anterior foi mantido e após a instalação das 8 (oito), houve modificação nos arquivos das VMs, mas, mais uma vez, não é impossível inferir o que causou as modificações dentro do sistema.

- **CHKRootkit:**
  O tempo de execução dos testes, foi de 9 (nove) à 25 (vinte e cinco) segundos, um tempo bem curto, porém como o CHKRootkit executa somente em *userland*, ele só irá verificar os arquivos do usuário e isso justificaria o tempo curto. O resultado da verificação anterior também foi mantido, nenhuma das 8 (oito) amostras foi encontrada.

- **ClamAV:**
  O tempo de execução ficou entre 15 (quinze) e 25 (vinte e cinco) minutos. O resultado das verificações foi o mesmo obtido no primeiro teste para 7 (sete) amostras. Para a amostra **m0nad** o ClamAV não conseguiu terminar a verificação e acabou terminando a sua execução precocemente devido a um erro que não foi identificado, mas acredita-se que ele tenha sido causado pela indisponibilidade do programa conseguir verificar os módulos binários desse *rootkit*, outra hipótese é que o próprio *rootkit* possa ter feito com que o programa sofresse esse "aborto" de execução ao perceber que estavam tentando analisar seus binários, mas não foi possível confirmar.

- **Lynis:**
  Os tempos de execução variaram entre 25 (vinte e cinco) segundos e 1 (um) minuto e 16 (dezesseis) segundos, foi considerado um tempo curto, mas resultados apresentaram algumas diferenças para o primeiro teste. Em alguns módulos foram apresentados erros de alguns arquivos que não puderam ser encontrados ou escaneados. Isto ocorreu para todas as 8 (oito) amostras.

- **OSSEC:**
  Não possível avaliar o tempo, pois o OSSEC analisa o Sistema Operacional em tempo real. Assim como o AIDE, seus alertas mostraram que houve modificação no sistema, graças ao verificador de integridade que possuí, mas nenhum alerta de *rootkit* foi identificado, acredita-se isto aconteceu porque, segundo pesquisas feitas no material dos desenvolvedores, entendeu-se que a presença dos *rootkits* é verificada baseado em uma base de assinaturas que o programa contém, como nenhuma das amostras tem um traço parecido, elas não foram identificadas.

- **RKHunter:**
  Os tempos foram de 1 (um) minuto e 10 (dez) segundos a 3 (três) minutos e 13 (treze) segundos. Este período de duração foi considerado curto, mas foi o programa que mais apresentou diferenças em seu resultado. O RKHunter não identificou as amostras como sendo *rootkits*, mas não só identificou algumas mudanças em suas análises, como também listou arquivos suspeitos, que variaram, em quantidade, entre 2 (dois) a 4 (quatro) para todas as amostras. Embora tenha sido apresentado essa diferença, não foi possível identificar quais arquivos eram os suspeitos.

- **Tripwire:**
  Os tempos variaram entre 1 (um) minuto e 5 (cinco) segundos a 2 (dois) minutos e 35 (trinta e cinco) segundos, uma variação curta para uma análise completa do sistema. Não foram obtidas diferenças em 7 (sete) das 8 (oito) verificações, somente a amostra **miagilepner** apresentou cerca de 368 (trezentos e sessenta e oito) erros devido a falta de imagens do sistema, que este exemplar deletou.

6.3.2.1 Avaliação dos Resultados

Os programas foram re-avaliados para os resultados da Segunda Fase de Teste. Na Tabela 6.2, foram colocadas todas as amostras na vertical e os programas na horizontal, e feito um cruzamento de em quais delas foram encontradas divergências no sistema. A avaliação dos programa foram feitas seguindo o mesmo padrão definido na **Seção 6.3.1.1**, e esta contida na Tabela 6.3.

Amostra/Programa	AIDE	CHKRootkit	ClamAV	Lynis	Ossec	RKHunter	Tripwire
cloudsec	✓	-	-	✓	✓	✓	-
leurfete	✓	-	-	✓	✓	✓	-
m0nad	✓	-	Crash	✓	✓	✓	-
miagilepner	✓	-	-	✓	✓	✓	✓
NoviceLive	✓	-	-	✓	✓	✓	-
QuokkaLight	✓	-	-	✓	✓	✓	-
t0t3m	✓	-	-	✓	✓	✓	-
zoeyqi	✓	-	-	✓	✓	✓	-

Tabela 6.2: Tabela de comparação dos programas em função do numero de alterações/arquivos suspeitos .

Programa	Qtde de Diferenças	Resposta da Análise	Tempo
AIDE	8	-	***
CHKRootkit	0	*	*
ClamAV	0	**	**
Lynis	8	-	*
Ossec	8	**	-
RKHunter	8	***	*
Tripwire	1	-	*

Tabela 6.3: Tabela de comparação dos programas avaliados segundo os resultados do segundo teste.

6.4 Conclusão

Os programas de detecção foram avaliados de diversas maneiras, sendo por material disponível, site do(s) desenvolvedor(es), forma de apresentação de resultados, usuários que poderiam utiliza-los, tempo e quantidade de diferenças identificadas. Porém, eles deixaram a desejar em quase todos os quesitos de avaliação.

Era esperado que o CHKRootkit, ClamAV e OSSEC não conseguissem identificar as amostras, ou por serem de *kernel*, ou por não estarem em seu banco de assinaturas, ou por não possuírem uma lógica capaz de localizar essas amostras, mas mesmo assim o tempo de verificação, curto na grande maioria dos casos, faz transparecer que o sistema não esta sendo verificado com todo o cuidado e em sua totalidade, e a forma com a qual os resultados foram apresentados demonstra certa falta de preocupação de como o usuário vai conseguir interpretar o que é escrito na tela ou o *log* gerado, pois não são informações de fácil identificação e leitura, demanda tempo e muito conhecimento, que mesmos os usuários com muita experiencia podem não saber como compreender.

Porém, a falha mais grave identificada é que somente 2 (dois) dos 7 (sete) programas avaliados possuem a capacidade de verificar sistemas externos, que são o ClamAV e o CHKRootkit, nenhum dos outros apresenta esta funcionalidade, o que atualmente não poderia acontecer, principalmente devido a capacidade de virtualização e ameaças diferentes que são identificadas todos os dias. Um *log* de um sistema infectado não pode ser considerado correto até que uma verificação externa seja feita e de o mesmo resultado, pois a capacidade de modificar e manipular dos *malwares* é constantemente relatada.

O resultado refletiu o quanto os programas para sistemas Unix ainda detêm um padrão antigo, onde os usuários conseguiam interpretar informações de baixo nível, atualmente não é mais desta maneira. Os computadores evoluíram de uma forma que não é mais necessário que o usuário comum veja a este nível tão baixo, e a grande maioria não sabe como interpretar isso, exigindo dos programadores uma interface mais amigável ou resultados compreensíveis, então é necessário que seja revisto e os softwares seja remodelados, mas não somente isso, é extremamente urgente que a lógica de detecção e o modo de verificação sejam repensados, pois existem ameaças perigosas que são multi-plataforma ou existem para sistemas Unix, talvez desenvolver técnicas em conjunto com sistemas bio-inspirados, ou tentar usar outras heurísticas, sejam opções possíveis para ajudar a começar a resolver o problema.

Capítulo 7

Conclusão e Trabalhos Futuros

Todos os programas avaliados foram submetidos a baterias individuais de análise e avaliações mediante as respostas apresentadas. Entretanto, depois de todo esse período de análises, chega-se à conclusão que os softwares disponíveis para sistema Linux não estão preparados para responder à ameaça dos *rootkits*, isto se deve principalmente a uma falha gravíssima encontrada que é a impossibilidade da maioria deles de permitir uma verificação de um sistema externo ao local onde o programa está instalado. O *rootkit* é principalmente reconhecido pela sua capacidade de modificar arquivos sensíveis que possibilitam a sua detecção, uma vez que a análise é feita em um sistema comprometido pode ser igualmente alterada por ele [*rootkit*]. O risco disso acontecer é extremamente alto, por isso o resultado de uma verificação feita nessas condições, não pode ser considerado válido.

Outro ponto considerado como negativo foi em relação aos materiais disponibilizados pelos desenvolvedores, alguns deles não tinham a quantidade necessária de informação, ou não apresentavam um conteúdo que fosse pontual para dúvidas básicas, como instalação e utilização, ou não apresentavam respostas para erros de instalação, *update* ou usabilidade.

O terceiro e último ponto negativo a ser abordado é em relação a como os resultados dos programas são apresentados aos usuários. Muitas vezes, o que o usuário deseja é ver na tela, sendo ela gráfica ou via terminal, algo escrito de forma simples e que ele consiga compreender que o seu sistema pode estar comprometido. Mesmo para aqueles que tem experiência, algo que necessite de muita dedicação e tempo para ser compreendido acaba abrindo margem para um interpretação errada e levando a falsos negativos. Quando uma análise é feita e surgem preguntas em vez de respostas, quer dizer que o modo de demostrar ao utilizador final o que o programa fez e o que ele buscou, está errado. Programas de detecção, em um modo geral, são construídos para responder a pergunta se existe ou não uma ameaça dentro do sistema, e ele deve ser capaz de traduzir a sua avaliação de forma que a resposta possa ser compreendida por qualquer tipo de usuário.

Após todo este trabalho de análise, a pergunta que precisa ser respondida é: como fazer a detecção de *rootkits*. Dos tipos de verificação observados, o de melhor desempenho, foi o de detecção por integridade, pois ele captura qualquer modificação do sistema, porém isso gera uma pergunta: quem fez a modificação? Não é possível saber se ela foi ocasionada de alguma atualização do sistema ou se foi um *malware*. Se fosse possível um modo de se ter certeza deste software descobrir quem fez a modificação, estes seriam aliados cruciais para a descoberta dos *rootkits* dentro do SO. Não é possível depender dos softwares de assinatura, pois caso seja feita uma mudança mínima, o *rootkit* deixará de ser reconhecido. Talvez, seja o momento de repensar as técnicas que são utilizadas, mas não só elas, mas também os Sistemas Operacionais de alguma

forma que os tornem mais resistentes a invasões externas e não dependam tanto da capacidade de prevenção do usuário.

Talvez seja o momento de investir nos estudos de técnicas bio-inspiradas, fazer do sistemas de *anti-malware* ferramentas semelhantes ao sistema imunológico humano, capaz de aprender a cada invasão e se tornar cada vez mais imune a cada ameaças. Ou repensar as heurísticas utilizadas, compreendendo as falhas dos algoritmos e tentar resolvê-las. Outro ponto seriam os desenvolvedores compreenderem o SO em sua totalidade para prever falhas e formalizar programas que podem arrumá-las ou impedir que sejam usadas, aumentando a prevenção por parte do Sistema Operacional.

No campo da segurança computacional existem inúmeros trabalhos que podem ser desenvolvidos, mas nesta área em especial seriam de interesse: (1) tentar saber o que ocasionou o hiato tanto do desenvolvimento de *rootkits* quanto de softwares de detecção; (2) Desenvolver uma análise de nível de ameaça de *rootkits* de firmware/hardware e virtualizados; (3) Estudo e desenvolvimento de softwares capazes de verificar a presença de *rootkits* em sistemas externos; (4) Fazer um estudos de novos métodos de análise envolvendo técnicas baseadas em Sistemas Imunológicos Artificiais ou outras técnicas Bio-Inspiradas; (5) Fazer estudos sobre ferramentas de outros SO e tentar aplicá-las para Linux; (6) Tentar identificar pontos de facilidade para a instalação de *malwares* no sistema Linux; (7) Estudar os tipos mais recentes de *rootkits* para sistemas Unix e identificar as falhas por onde eles se instalam e dar possíveis soluções para os problemas; (8) Fazer um estudo com avaliativo, utilizando usuários comuns para fazer uma analise de opinião sobre os softwares de detecção.

Referências Bibliográficas

[Arciryas, 2016] Arciryas (2016). Rootkit sample code. https://github.com/Arciryas/rootkit-sample-code. Acessado em 22/10/2016.

[ARIS-LD, 2012] ARIS-LD (2012). Rootkit brasileiro tenta se esconder de antivírus e redireciona bancos. http://www.linhadefensiva.org/2012/11/rootkit-brasileiro-tenta-se-esconder-de-antivirus-e-redireciona-bancos/. Acessado em 04/08/2016.

[Audit, 2016] Audit, L. (2016). Projects - security auditing tool - rootkit.nl. https://rootkit.nl/projects/. Acessado em 02/08/2016.

[Bitdefender, 2016] Bitdefender (2016). Software anti-malware - bitdefender total security 2016. http://www.bitdefender.com.br/solutions/total-security.html?sem_region=COM&utm_source=Google&utm_campaign=BR_Bitdefender_BrExPh&sem_type=search&sem_placement=&utm_content=101735673972&utm_term=bitdefender. Acessado em 02/08/2016.

[Boelen, 2014] Boelen, M. (2014). The rootkit hunter project. http://www.chkrootkit.org/. Acessado em 02/08/2016.

[Bunten, 2004] Bunten, A. (2004). Unix and linux based rootkits techniques and countermeasures. https://www.first.org/conference/2004/papers/c17.pdf. Acessado em 20/07/2016.

[Butler et al., 2006] Butler, J., Arbaugh, B. e Petroni, N. (2006). The exponential growth of rootkit techniques. http://www.blackhat.com/presentations/bh-usa-06/BH-US-06-Butler.pdf. Acessado em 20/07/2016.

[Coppola, 2014] Coppola, M. (2014). Suterusu. https://github.com/mncoppola/suterusu. Acessado em 22/10/2016.

[de Beauchêne, 2016] de Beauchêne, T. (2016). Toorkit. https://github.com/deb0ch/toorkit. Acessado em 22/10/2016.

[F-Secure, 2016] F-Secure (2016). Award-winning antivirus software for all your devices. https://www.f-secure.com/en/web/home_global/products. Acessado em 02/08/2016.

[Greg Hoglund, 2005] Greg Hoglund, J. B. (2005). *Rootkits: Subverting the Windows Kernel*. Addison-Wesley Professional.

[Hu, 2016] Hu, N. (2016). Rootkit. https://github.com/TengHu/Rootkit. Acessado em 22/10/2016.

[Ismail, 2014] Ismail, A. H. (2014). Linux rootkit. https://github.com/ah450/rootkit. Acessado em 22/10/2016.

[Kaspersky, 2016] Kaspersky (2016). Kaspersky lab | antivirus protection '|&' internet security software. http://brazil.kaspersky.com/?domain=kaspersky.com. Acessado em 02/08/2016.

[King et al., 2006] King, S. T., Chen, P. M., Wang, Y.-M., Wang, H., Lorch, J. R. e Lorch, J. (2006). Subvirt: Implementing malware with virtual machines. https://www.microsoft.com/en-us/research/publication/subvirt-implementing-malware-with-virtual-machines. Acessado em 14/07/2016.

[Kleissner, 2007] Kleissner, P. (2007). Stoned bootkit. http://e.sebug.net/paper/Meeting-Documents/Blackhat-USA2009/BHUSA09-Kleissner-StonedBootkit-PAPER.pdf. Acessado em 20/07/2016.

[Kálnai, 2015] Kálnai, P. (2015). Linux ddos trojan hiding itself with an embedded rootkit. https://blog.avast.com/2015/01/06/linux-ddos-trojan-hiding-itself-with-an-embedded-rootkit/. Acessado em 14/07/2016.

[Kong, 2007] Kong, J. (2007). *Designing BSD Rootkits: An Introduction to Kernel Hacking*. No Starch Press.

[Kumar e Kumar, 2007] Kumar, N. e Kumar, V. (2007). Vbootkit: Compromising windows vista security. http://www.blackhat.com/presentations/bh-europe-07/Kumar/Presentation/bh-eu-07-kumar-apr19.pdf. Acessado em 14/07/2016.

[Lampis, 2016] Lampis, K. (2016). Muriel. https://github.com/klampworks/muriel. Acessado em 22/10/2016.

[Lehti et al., 2016] Lehti, R., Virolainen, P., den Berg, R. V. e von Haugwitz, H. (2016). Advanced intrusion detection environment. http://aide.sourceforge.net/. Acessado em 02/08/2016.

[Leyden, 2007] Leyden, J. (2007). Greek mobile wiretap scandal unpicked. http://www.theregister.co.uk/2007/07/11/greek_mobile_wiretap_latest/. Acessado em 20/07/2016.

[Light, 2016] Light, Q. (2016). rkduck - rootkit for linux v4. https://github.com/QuokkaLight/rkduck. Acessado em 22/10/2016.

[Malwarebytes, 2016] Malwarebytes (2016). Malwarebytes anti-rootkit beta. https://www.malwarebytes.org/antirootkit/. Acessado em 02/08/2016.

[Manap, 2001] Manap, S. (2001). Rootkit: Attacker undercover tools. http://www.cybersecurity.my/data/content_files/13/82.pdf. Acessado em 20/07/2016.

[Mello, 2016] Mello, V. R. (2016). Diamorphine. https://github.com/m0nad/Diamorphine. Acessado em 22/10/2016.

[miagilepner, 2016] miagilepner (2016). Porny. https://github.com/miagilepner/porny. Acessado em 22/10/2016.

[Murilo e Steding-Jessen, 2014] Murilo, N. e Steding-Jessen, K. (2014). chkrootkit – locally checks for signs of a rootkit. http://www.chkrootkit.org/. Acessado em 02/08/2016.

[nsarlin, 2014] nsarlin (2014). Backdoorlinux. https://github.com/nsarlin/BackdoorLinux. Acessado em 22/10/2016.

[nu11secur1ty, 2016] nu11secur1ty (2016). example-rootkit. https://github.com/nu11secur1ty/example-rootkit. Acessado em 22/10/2016.

[Odu, 0016] Odu, D. (20016). Linux rootkit. https://github.com/Davodu/Linux-Rootkit. Acessado em 22/10/2016.

[of Health, 2016] of Health, N. N. I. (2016). Competencies proficiency scale. https://hr.od.nih.gov/workingatnih/competencies/proficiencyscale.htm. Acessado em 17/12/2016.

[Oliveira, 2008] Oliveira, C. (2008). Rootkit: Uma nova ameaça? https://www.vivaolinux.com.br/artigo/Rootkit-Uma-nova-ameaca?pagina=1. Acessado em 15/10/2016.

[Phillips, 2016] Phillips, M. (2016). Simple rootkit. https://github.com/leurfete/simple-rootkit. Acessado em 22/10/2016.

[Rohr, 2005] Rohr, A. (2005). Rootkit é usado para trapaça em games. http://www.linhadefensiva.org/2005/11/rootkit-cheat-games/. Acessado em 02/08/2016.

[Rohr, 2006a] Rohr, A. (2006a). Pesquisador demonstra rootkits em placas pci. http://www.linhadefensiva.org/2006/11/pci-rootkit/. Acessado em 02/08/2016.

[Rohr, 2006b] Rohr, A. (2006b). Tecnologias rootkit em softwares de segurança. http://www.linhadefensiva.org/2006/01/rootkit-software-sec/. Acessado em 04/08/2016.

[Rutkowska e Tereshkin, 2006] Rutkowska, J. e Tereshkin, A. (2006). Bluepilling the xen hypervisor. http://lira.epac.to/DOCS-TECH/Hacking/Bluepilling%20the%20Xen%20Hypervisor.pdf. Acessado em 20/07/2016.

[Shevchenko, 2008] Shevchenko, A. (2008). Rootkit evolution. https://securelist.com/analysis/publications/36222/rootkit-evolution/. Acessado em 14/07/2016.

[Simioni, 2015] Simioni, D. (2015). Jellyfish: Novo rootkit pode afetar também o linux. http://www.diolinux.com.br/2015/05/jellyfish-novo-rootkit-pode-afetar-linux.html. Acessado em 02/08/2016.

[Simpson et al., 2012] Simpson, M. T., Backman, K. e Corley, J. (2012). *Hands-On Ethical Hacking and Network Defense 1st edition*. Course Technology.

[soad003, 2016] soad003 (2016). rootkit. https://github.com/soad003/rootkit. Acessado em 22/10/2016.

[Soeder e Permeh, 2005] Soeder, D. e Permeh, R. (2005). eeye bootroot. https://www.blackhat.com/presentations/bh-usa-05/bh-us-05-soeder.pdf. Acessado em 14/07/2016.

[Sokolov, 2016] Sokolov, D. (2016). Unhackme. http://greatis.com/unhackme/. Acessado em 02/08/2016.

[Swimmer, 2008] Swimmer, M. G. (2008). *Malware Intrusion Detection*. Owlfin, LLC.

[Sysinternals, 2016] Sysinternals, W. (2016). Tn rootkit revealer. https://technet.microsoft.com/en-us/sysinternals/bb897445.aspx. Acessado em 02/08/2016.

[t0t3m, 2014] t0t3m (2014). Afkit. https://github.com/t0t3m/AFkit. Acessado em 22/10/2016.

[TEHN, 2005] TEHN (2005). The basics of rootkits: Leave no trace. https://www.ethicalhacker.net/features/book-reviews/the-basics-of-rootkits-leave-no-trace. Acessado em 20/07/2016.

[TrendMicro, 2016a] TrendMicro (2016a). Pokémon-themed umbreon linux rootkit hits x86, arm systems. http://blog.trendmicro.com/trendlabs-security-intelligence/pokemon-themed-umbreon-linux-rootkit-hits-x86-arm-systems/. Acessado em 14/07/2016.

[TrendMicro, 2016b] TrendMicro (2016b). Trend micro software download center. http://downloadcenter.trendmicro.com/index.php?regs=NABU&clk=latest&clkval=355&lang_loc=1. Acessado em 02/08/2016.

[Tripwire, 2016] Tripwire, I. (2016). Open source tripwire. https://sourceforge.net/projects/tripwire/. Acessado em 02/08/2016.

[Wikipedia, 2016a] Wikipedia (2016a). Open source tripwire. https://en.m.wikipedia.org/wiki/Open_Source_Tripwire. Acessado em 20/07/2016.

[Wikipedia, 2016b] Wikipedia (2016b). Rootkit. https://en.wikipedia.org/wiki/Rootkit. Acessado em 20/07/2016.

[wzt, 2015] wzt (2015). Brootkit. https://github.com/cloudsec/brootkit. Acessado em 22/10/2016.

[Xu, 2016] Xu, Q. (2016). Rootkit-functionality-implementation-under-unix. https://github.com/xuqicx23/Rootkit-functionality-implementation-under-Unix. Acessado em 22/10/2016.

[Zhengxiong, 2016] Zhengxiong, G. (2016). Libzeroevil and the research rootkit project. https://github.com/NoviceLive/research-rootkit. Acessado em 22/10/2016.

[zoeyqi, 2016] zoeyqi (2016). Rootkit. https://github.com/zoeyqi/Rootkit. Acessado em 22/10/2016.

[Zovi, 2006] Zovi, D. A. D. (2006). Hardware virtualization rootkits. http://www.blackhat.com/presentations/bh-usa-06/BH-US-06-Zovi.pdf. Acessado em 14/07/2016.

Apêndice A

Documentação dos Rootkits

A.1 Brootkit

Esta é a documentação fornecida pelo autor[wzt, 2015]:

"**BROOTKIT**

Lightweight rootkit implemented using bash shell scripts v0.10

by wzt 2015 wzt.wzt@gmail.com

If bash shell scripts can be designed for security tools like chkrootkit or rkhunter, so too can it be implemented for a rootkit.

FEATURES

1. more hidable ability against admintrator or hids.
2. su passwd thief.
3. hide file and directories.
4. hide process.
5. hide network connections.
6. connect backdoor.
7. multi thread port scanner.
8. http download.
9. multi thread ssh passwd crack.

TARGET OS

1. centos
2. rhel
3. ununtu
4. debina
5. fedora

6. freebsd

TODO

1. sudo thief support.

INSTALL

Linux distribution systems.

1. edit br.conf first

```
#these ports will be hidden: port1,port2,...,portn.
HIDE_PORT         8080,8899
#these files will be hidden: file1,file2,...,filen.
HIDE_FILE         br.conf,bashbd.sh,brootkit,.bdrc,
   brdaemon
#these processes will be hidden: process1,process2,...,
   processn.
HIDE_PROC         bashbd,brootkit,pty.spawn,brdaemon
#the connect back host domain name or ip address.
REMOTE_HOST       localhost
#the connect back host port.
REMOTE_PORT       8080
#the connect backdoor base sleep time.
SLEEP_TIME        60
```

2. ./install.sh
3. multi thread port scanner.

```
[root@localhost brootkit]{cifrao} ./brscan.sh
./brscan.sh <-p> [-n|-t|-o|-h] <remote\_host>
```

Option:

- -p ports, pattern: port1,port2,port3-port7,portn...
- -n thread num, default is 10
- -t timeout, default is 30s
- -o results write into log file, default is brscan.log
- -h help information.

Exp:

```
./brscan.sh -p 21,22,23-25,80,135-139,8080 -t 20 www.
   cloud-sec.org
./brscan.sh -p 1-65525 -n 200 -t 20 www.cloud-sec.org
```

```
[root@localhost brootkit]# ./brscan.sh -p
   21,22,23-25,80,135-139,8080 -t 5 -n 20 www.wooyun.
   org
host: www.wooyun.org | total ports: 10 | thread num: 10
    timeout: 5 | logfile: brscan.log

thread<0 >        --          pid <57053> -->   21
thread<1 >        --          pid <57054> -->   22
```

```
6       thread<2 >        --          pid <57055> -->   23
7       thread<3 >        --          pid <57056> -->   24
8       thread<4 >        --          pid <57057> -->   80
9       thread<5 >        --          pid <57058> -->   135
10      thread<6 >        --          pid <57059> -->   136
11      thread<7 >        --          pid <57060> -->   137
12      thread<8 >        --          pid <57061> -->   138
13      thread<9 >        --          pid <57070> -->   8080
14
15      [>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>] 10/10
            6 s
16
17      www.wooyun.org: 80
```

4. multi thread ssh passwd crack.

```
1       [root@localhost brootkit]# ./sshcrack.sh
2       ./sshcrack.sh <-h host> <-u user> <-p passwd> [-t
           timeout] [-n threadnum] [-o logfile]
```

Option:

- -h host name or host list file.
- -u user name or user list file.
- -p single passwd or passwd list file.
- -t connect timeout, defalut is 5s.
- -n thread num, default is 1.
- -o log file.
- -v display help information.

exp:

```
1       ./sshcrack.sh -h 192.168.215.148 -u wzt -p passwd.lst
2       ./sshcrack.sh -h 192.168.215.148 -u wzt -p passwd.lst -
           n 10 -t 2
3       ./sshcrack.sh -h 192.168.215.148 -u user.lst -p passwd.
           lst -n 10 -t 2
4       ./sshcrack.sh -h host.lst -u user.lst -p passwd.lst -n
           10 -t 2
```

```
1       [root@localhost brootkit]# ./sshcrack.sh -h
           192.168.215.148 -u wzt -p passwd.lst -n 6
2       host: 1 | users: 1 | passwd: 28 thread: 6 | timeout: 10
            | logfile: sshcrack.log
3
4       Thread[ 1]  wzt@192.168.215.148     ==>   [e           ]
                [failed]     3
5       Thread[ 2]  wzt@192.168.215.148     ==>   [a           ]
                [failed]     3
6       Thread[ 3]  wzt@192.168.215.148     ==>   [d           ]
                [failed]     3
7       Thread[ 4]  wzt@192.168.215.148     ==>   [giveshell   ]
                [success]    6
8       Thread[ 5]  wzt@192.168.215.148     ==>   [123456      ]
                [failed]     3
9       Thread[ 6]  wzt@192.168.215.148     ==>   [fd          ]
                [failed]     3
```

```
10
11      waiting all threads to finsh...
```

Freebsd system
on the modern freebsd system root use csh by default, the other users use sh default. this version of brootkit can only support sh based features.

1. edit brsh.conf first

```
1       #this port will be hidden.
2       HIDE_PORT         8080
3       #these files will be hidden.
4       HIDE_FILE         brsh
5       #these process will be hidden.
6       HIDE_PROC         sh
7       #the connect back host domain name or ip address.
8       REMOTE_HOST       localhost
9       #the connect back host port.
10      REMOTE_PORT       8080
11      #the connect backdoor base sleep time.
12      SLEEP_TIME        60
```

   brshootkit config file, only one argument support.

2. ./install.sh

SOURCE

https://github.com/cloudsec/brootkit" (cloudsec - wzt)

A.2 leurfete

Esta é a documentação fornecida pelo autor[Phillips, 2016]:

"**Simple Rootkit**

A simple attack via kernel module, with highly detailed comments.

Here we'll compile a kernel module which intercepts every "read" system call, searches for a string and replaces it if it looks like the gcc compiler or the python interpreter. This is meant to demonstrate how a compromised system can build a malicious binary from perfectly safe source code.

For more information see: http://mrrrgn.com/18/

Also check out: http://memset.wordpress.com/2010/12/03/syscall-hijacking-kernel-2-6-systems/

Instructions

Install your kernel headers

```
1       sudo apt-get install linux-headers-{cifrao}(uname -r)
```

Run make

```
cd simple-rootkit && make
```

Load the module

```
sudo insmod simple-rootkit.ko
```

Compile any C or run any Python script and all instances of the string “World!” will now read as Mrrrgn.

```
gcc hello.c -o hello
./hello
```

” (Morgan Phillips)

A.3 m0nad

Esta é a documentação fornecida pelo autor[Mello, 2016]:

“**Diamorphine**

Diamorphine is a LKM rootkit for Linux Kernels 2.6.x/3.x/4.x Features

- When loaded, the module starts invisible;
- Hide/unhide any process by sending a signal 31;
- Sending a signal 63(to any pid) makes the module become (in)visible;
- Sending a signal 64(to any pid) makes the given user become root;
- Files or directories starting with the MAGIC_PREFIX become invisble;
- Source: https://github.com/m0nad/Diamorphine

Install

Verify if the kernel is 2.6.x/3.x/4.x

```
uname -r
```

Clone the repository

```
git clone https://github.com/m0nad/Diamorphine
```

Enter the folder

```
cd Diamorphine
```

Compile

```
make
```

Load the module(as root)

```
insmod diamorphine.ko
```

Uninstall

The module starts invisible, to remove you need to make its visible

```
kill -63 0
```

Then remove the module(as root)

```
rmmod diamorphine
```

References

Wikipedia Rootkit https://en.wikipedia.org/wiki/Rootkit

Linux Device Drivers http://lwn.net/Kernel/LDD3/

LKM HACKING https://www.thc.org/papers/LKM_HACKING.html

Memset's blog http://memset.wordpress.com/

Linux on-the-fly kernel patching without LKM http://phrack.org/issues/58/7.html

WRITING A SIMPLE ROOTKIT FOR LINUX http://big-daddy.fr/repository/Documentation/Hacking/Security/Malware/Rootkits/writing-rootkit.txt

Linux Cross Reference http://lxr.free-electrons.com/" (Victor Ramos Mello)

A.4 mncoppola

Esta é a documentação fornecida pelo autor[Coppola, 2014]:

"**Suterusu**

Typical compilation steps:

```
wget http://kernel.org/linux-x.x.x.tar.gz
tar xvf linux-x.x.x.tar.gz
cd linux-x.x.x
make menuconfig
make modules\_prepare
cd /path/to/suterusu
make linux-x86 KDIR=/path/to/kernel
```

To compile against the currently running kernel (kernel headers installed):

```
make linux-x86 KDIR=/lib/modules/{cifrao}(uname -r)/build
```

If a specific toolchain is desired for cross-compilation, provide the CROSS_COMPILE variable during make:

```
make android-arm CROSS\_COMPILE=arm-linux-androideabi- KDIR=/path/
   to/kernel
```

To compile the command binary:

```
gcc sock.c -o sock
```

Commands

Root shell

```
./sock 0
```

Hide PID

```
./sock 1 [pid]
```

Unhide PID

```
./sock 2 [pid]
```

Hide TCPv4 port

```
./sock 3 [port]
```

Unhide TCPv4 port

```
./sock 4 [port]
```

Hide TCPv6 port

```
./sock 5 [port]
```

Unhide TCPv6 port

```
./sock 6 [port]
```

Hide UDPv4 port

```
./sock 7 [port]
```

Unhide UDPv4 port

```
./sock 8 [port]
```

Hide UDPv6 port

```
./sock 9 [port]
```

Unhide UDPv6 port

```
./sock 10 [port]
```

Hide file/directory

```
./sock 11 [name]
```

Unhide file/directory

```
./sock 12 [name]
```

Hide network PROMISC flag

```
./sock 13
```

Unhide network PROMISC flag

```
./sock 14
```

Enable module loading (force kernel.modules_disabled=0)

```
./sock 15
```

Silently prohibit module loading (neutralize future loaded modules)

```
./sock 16
```

Silently re-permit module loading (undo command 16)

```
./sock 17
```

File/directory hiding

At the moment, file/dir hiding only hides names on the / filesystem. Note that names are hidden, not paths. For instance, giving the name ".blah" to Suterusu will hide the name ".blah" in all directories on the filesystem." (Michael Coppola)

A.5 NoviceLive

Esta é a documentação fornecida pelo autor[Zhengxiong, 2016]:

"**LibZeroEvil & the Research Rootkit project**

This is LibZeroEvil & the Research Rootkit project, in which there are step-by-step, experiment-based courses that help to get you started and keep your hands dirty with offensive or defensive development in the Linux kernel, and also guide you with demonstrative examples through the underlying core library, LibZeroEvil, which attempts to be a real-world consumable programming framework for any evil or good kernel-land invaders or defenders.

That being said, it's just the beginning and LibZeroEvil is still in its infancy, serving educational purposes mainly. Warning

It's never recommended to perform kernel module experiments on a physical machine, unless the owner will never complain about frequent rebooting or forced halting and possible data or work loss.

You have been warned. Guidelines on Creating New Issues or Contributing

If a course doesn't compile, or do compile but doesn't work as expected, e.g. crashing or hanging your system, feel free to create a new issue. But before that, consider the following.

Search existing issues to ensure that it won't be duplicated.

Attach detailed information of your system, e.g., uname –all, and what your compiler throws on you, i.e., the error information, so that others can successfully reproduce your issue and manage to help you out.

If you can't figure out which error is the most significant one, paste all of them verbatim inside Markdown triple quotes.

Remember that kernel compatibility issues are the most common ones, since the author paid virtually no attention to that. However, it's my pleasure to learn about and fix them.

Some courses are x64 only for the time being.

Information on Kernel Compatibility Tested Kernel

My major development environment.

Kali

Linux anon 4.6.0-kali1-amd64 #1 SMP Debian 4.6.3-1kali1 (2016-07-12) x86_64 GNU/Linux.

Compilable Kernel

That is, ./tests/makeall.sh –quiet reports no error, but I haven't tested the functionality.

Arch

Linux anon 4.6.4-1-ARCH #1 SMP PREEMPT Mon Jul 11 19:12:32 CEST 2016 x86_64 GNU/Linux.

Ubuntu 14.04

Linux anon 4.2.0-42-generic #49 14.04.1-Ubuntu SMP Wed Jun 29 20:22:11 UTC 2016 x86_64 x86_64 x86_64 GNU/Linux

Known Unresolved Compatibility Issues

struct dir_context doesn't exist on kernel version 3.10 and earlier.

For example, CentOS 7, Linux localhost.localdomain 3.10.0-327.22.2.el7.x86_64 #1 SMP Thu Jun 23 17:05:11 UTC 2016 x86_64 x86_64 x86_64 GNU/Linux.

Notice on Directory Structure

The directory structure of this repository might change drastically without any notifications. Available Courses

More courses might be designed and added here later.

- Course 1: Modifying / Hooking the sys_call_table
  - Experiment 1: hello
    Hello World! kernel module.
  - Experiment 2: sys_call_table
    Get sys_call_table 's address by brute-force memory searching starting from PAGE_OFFSET.
  - Experiment 3: write_protection
    Disable or enable Write Protection via the CR0 register.
  - Roundup Experiment 1: fsmon
    A primitive file monitor based on system call hooking.
    Hooked functions: open, unlink, unlinkat.
  - Roundup Experiment 2: psmon
    A primitive process monitor via system call hooking.
    Hooked functions: execve. Notice
    I have seen reports that the method used in this experiment would not work normally due to inconsistent ABI of stub_execve, which requires further investigation.

  - Roundup Experiment 3: fshid
    A primitive file-hiding demonstration using system call hooking.
    Hooked functions: getdents, getdents64.
    Hidden files: 032416_525.mp4.

- Course 2: Implementing fundamental functionalities of rootkits
  - Experiment 1: root
    Providing a root backdoor.
  - Experiment 2: komon
    Preventing modules from initializing and functioning by substituting their init and exit functions when MODULE_STATE_COMING is notified to module notifiers.
  - Experiment 3: fshid
    Hiding files by hooking filldir.
  - Experiment 4: pshid
    Hiding processes by hiding entries under /proc.
  - Experiment 5: pthid
    Hiding ports by filtering contents in /proc/net/tcp and the like by hooking the show function of their seq_file interfaces.
  - Experiment 6: kohid
    Hiding modules by hiding entries in /sys/module and filtering contents of /proc/modules by hooking its show function.
    This experiment combines the techniques demonstrated in Experiment 4: pshid and Experiment 5: pthid.

- Course 3: Infecting critical kernel modules for persistence and more
  - Experiment 1: elf
    Providing elementary materials on ELF parsing and modifying.
    This experiment implemented an essential tool, i.e. setsym, for following experiments, and also two trivial tools, lssec resembling readelf -S and lssym resembling readelf -s / objdump -t.
    They are coded for 64-bit ELF only, but it shouldn't be difficult to adapt.
  - Experiment 2: noinj
    Hijacking / Hooking the init and exit function of the module with functions in the same module by modifying the symbol table.
  - Experiment 3: codeinj
    Injecting the adapted fshid (See Experiment 3 of Course 2) into a demonstrative simple module (i.e. without static __init or static __exit), and hooking / hijacking its init and exit functions by modifying the symbol table.
  - Roundup Experiment: real
    Injecting the adapted fshid (See Experiment 3 of Course 2) into a real-world kernel module (i.e. with static __init or static __exit) by linking, and hooking / hijacking its init and exit functions by modifying the symbol table.

- Course 4: Modifying / Patching the entry_SYSCALL_64
  This is x64 only. However, it's not difficult to adapt.

  - Experiment 1: get
    Getting the sys_call_table 's address in the machine code of entry_SYSCALL_64 by searching the identifying bytes ff 14 c5.
  - Experiment 2: set
    Patching the sys_call_table 's address in the machine code of entry_SYSCALL_64 with a faked but innocuous, i.e. unmodified, one.
  - Experiment 3: rec
    Recovering the sys_call_table 's address in the machine code of entry_SYSCALL_64 to that obtained via sys_close -based memory searching.
  - Roundup Experiment: ifmon
    Monitoring network flow (especially GET & POST) by hooking sys_sendto using the method demonstrated in the above three experiments on entry_SYSCAL_64.
- Course 5: Inline Hooking
  - Experiment 1: jmp
    Patching the starting bytes of target functions with control flow redirection instructions, e.g. PUSH RET, JMP or INT, which transfer control to our function, where our tasks are performed, including restoring those bytes and invoking the victim function if necessary.

Projects Of Interests

- mncoppola/suterusu
  An LKM rootkit targeting Linux 2.6/3.x on x86(_64), and ARM.
- maK-/maK_it-Linux-Rootkit
  This is a linux rootkit using many of the techniques described on http://r00tkit.me.
- ivyl/rootkit
  Sample Rootkit for Linux.
- cloudsec/brootkit
  Lightweight rootkit implemented by bash shell scripts v0.10.

References & Further Readings

Suterusu Rootkit: Inline Kernel Function Hooking on x86 and ARM Infecting loadable kernel modules: kernel versions 2.6.x/3.0.x"

(Gu Zhengxiong)

A.6 nu11secur1ty

Esta é a documentação fornecida pelo autor[nu11secur1ty, 2016]:

" **Example Rootkit**

A simple attack via kernel module, with highly detailed comments.

Here we'll compile a kernel module which intercepts every "read" system call, searches for a string and replaces it if it looks like the gcc compiler or the python

interpreter. This is meant to demonstrate how a compromised system can build a malicious binary from perfectly safe source code.

For more information see: http://mrrrgn.com/18/

Also check out: http://memset.wordpress.com/2010/12/03/syscall-hijacking-kernel-2-6-systems/

Instructions

Install your kernel headers

```
sudo apt-get install linux-headers-{cifrao}(uname -r)
```

Run make

```
cd simple-rootkit \&\& make
```

Load the module

```
sudo insmod simple-rootkit.ko
```

Compile any C or run any Python script and all instances of the string "World!" will now read as Mrrrgn.

```
gcc hello.c -o hello
./hello
```

" (nu11secur1ty)

A.7 QuokkaLight

Esta é a documentação fornecida pelo autor[Light, 2016]:

"**rkduck - Rootkit for Linux v4 Build Status**

rkduck is a Loadable Kernel Module rootkit for the latest Linux Kernels v4. This is still a work in progress.

Features

- Stealth
- Hide files, directories, processes
- Communication
- SSH
- Direct shell (unencrypted)
- Reverse shell (unencrypted)
- Keylogger
- Recording of the keystrokes of every user.
- Information sent periodically
- Crumbs

- A user space CLI program allowing the user to control the rootkit configuration during its execution
- Requires an authentication to be used (hardcoded key stored in rduck, the configuration section has more information about it)

Tests

At the moment we didn't get the chance to test our rootkit on different versions of Linux to make sure everything is working as intended. If you want to report a bug feel free to create an issue or send us an email at quokkalight@gmail.com. Contributors

mpgn - Twitter

RainbowLyte - Twitter" (Quokka Light)

A.8 t0t3m

Esta é a documentação fornecida pelo autor[t0t3m, 2014]:

"**AFkit**
Description

It is able to:

- Hide himself from /proc/modules, /proc/kallsyms and /sys/modules
- Hide files with "__rt" substring in their name (and their content)
- Avoid the opening and reading of /dev/mem, /dev/port and /dev/kmem devices
- This anti-forensic rootkit uses the system call hijacking method, in particular are hijacked the following syscalls:
  - open
  - read
  - getdents
  - getdents64
  - chdir
  - kill

ToDo

- Hide network connections
- Hide network ports
- Hide process by given PID

PLEASE REPORT BUGS. IT'LL BE VERY APPRECIATED!

Tested on ArchLinux Kernel 3.10.10 (x86_64) but it is supposed to work on all 3.x versions. Beta quality product. I don't take any responsability about its usage and its behaviour.

UPDATE 19/04/2014

Tested on ArchLinux with Kernel 3.14.1 (x86_64) and Debian Wheezy with kernel 3.12 (686)"

(t0t3m)

A.9 zoeyqi

Esta é a documentação fornecida pelo autor[zoeyqi, 2016]:

"**Rootkit**

1. Create a attack program and a Sneaky Kernel Module (a Linux Kernel Module – LKM) with the following functions: 1.It will hide the "sneaky_process" executable file from both the 'ls' and 'find' UNIX commands.
2. In a UNIX environment, every executing process will have a directory under /proc that is named with its process ID (e.g /proc/1480). This directory contains many details about the process. The sneaky kernel module will hide the /proc/ directory
3. It will hide the modifications to the /etc/passwd file that the sneaky_process made. It will do this by opening the saved "/tmp/passwd" when a request to open the "/etc/passwd" is seen.
4. It will hide the fact that the sneaky_module itself is an installed kernel module. "

(zoeyqi)